Autorização SaaS multilocatário e controle de acesso à API: opções de implementação e melhores práticas

AWS Orientação prescritiva

AWS Orientação prescritiva: Autorização SaaS multilocatário e controle de acesso à API: opções de implementação e melhores práticas

Table of Contents

Autorização SaaS multilocatário e controle de acesso à API: opções de implementação e melhores práticas

Tabby Ward, Thomas Davis, Gideon Landeman e Tomas Riha, da Amazon Web Services ()AWS

Maio de 2024 (histórico do documento)

A autorização e o controle de acesso à API são um desafio para muitos aplicativos de software — em particular, para aplicativos de software como serviço (SaaS) multilocatários. Essa complexidade é evidente quando você considera a proliferação de microsserviços APIs que devem ser protegidos e o grande número de condições de acesso que surgem de diferentes inquilinos, características do usuário e estados do aplicativo. Para resolver esses problemas de forma eficaz, uma solução deve impor o controle de acesso entre os muitos APIs apresentados por microsserviços, camadas de Backend for Frontend (BFF) e outros componentes de um aplicativo SaaS multilocatário. Essa abordagem deve ser acompanhada por um mecanismo capaz de tomar decisões de acesso complexas com base em muitos fatores e atributos.

Tradicionalmente, o controle de acesso e a autorização da API eram tratados pela lógica personalizada no código do aplicativo. Essa abordagem estava sujeita a erros e não era segura, pois os desenvolvedores que tinham acesso a esse código podiam alterar acidentalmente ou deliberadamente a lógica de autorização, o que poderia resultar em acesso não autorizado. Foi difícil auditar as decisões tomadas pela lógica personalizada no código do aplicativo, porque os auditores precisariam mergulhar na lógica personalizada para determinar sua eficácia na manutenção de qualquer padrão específico. Além disso, o controle de acesso à API geralmente era desnecessário, porque não havia tantos APIs para proteger. A mudança de paradigma no design de aplicativos para favorecer microsserviços e arquiteturas orientadas a serviços aumentou o número de pessoas APIs que precisam usar uma forma de autorização e controle de acesso. Além disso, a necessidade de manter o acesso baseado em inquilinos em um aplicativo SaaS multilocatário apresenta desafios adicionais de autorização para preservar a locação. As melhores práticas descritas neste guia oferecem vários benefícios:

- A lógica de autorização pode ser centralizada e escrita em uma linguagem declarativa de alto nível que não é específica de nenhuma linguagem de programação.
- A lógica de autorização é abstraída do código do aplicativo e pode ser aplicada como um padrão repetível a tudo APIs em um aplicativo.
- A abstração evita alterações acidentais dos desenvolvedores na lógica de autorização.

- A integração em um aplicativo SaaS é consistente e simples.
- A abstração evita a necessidade de escrever uma lógica de autorização personalizada para cada endpoint da API.
- As auditorias são simplificadas porque um auditor não precisa mais revisar o código para determinar as permissões.
- A abordagem descrita neste guia oferece suporte ao uso de vários paradigmas de controle de acesso, dependendo dos requisitos de uma organização.
- Essa abordagem de autorização e controle de acesso fornece uma maneira simples e direta de manter o isolamento dos dados do inquilino na camada de API em um aplicativo SaaS.
- As melhores práticas fornecem uma abordagem consistente para integrar e desembarcar inquilinos em relação à autorização.
- Essa abordagem oferece diferentes modelos de implantação de autorização (agrupados ou em silos), que têm vantagens e desvantagens, conforme discutido neste guia.

Resultados de negócios desejados

Esta orientação prescritiva descreve padrões de design repetíveis para controles de autorização e acesso à API que podem ser implementados para aplicativos SaaS multilocatários. Essa orientação é destinada a qualquer equipe que desenvolve aplicativos com requisitos de autorização complexos ou necessidades estritas de controle de acesso à API. A arquitetura detalha a criação de um ponto de decisão política (PDP) ou mecanismo de política e a integração de pontos de aplicação de políticas (PEP) em. APIs Duas opções específicas para criar uma PDP são discutidas: usar Amazon Verified Permissions com o Cedar SDK e usar o Open Policy Agent (OPA) com a linguagem de política Rego. O guia também discute a tomada de decisões de acesso com base em um modelo de controle de acesso baseado em atributos (ABAC) ou modelo de controle de acesso baseado em função (RBAC), ou uma combinação dos dois modelos. Recomendamos que você use os padrões e conceitos de design fornecidos neste guia para informar e padronizar sua implementação de autorização e controle de acesso à API em aplicativos SaaS multilocatários. Essa orientação ajuda a alcançar os seguintes resultados comerciais:

- Arquitetura padronizada de autorização de API para aplicativos SaaS multilocatários — Essa arquitetura distingue entre três componentes: o ponto de administração de políticas (PAP), onde as políticas são armazenadas e gerenciadas, o ponto de decisão de política (PDP), onde essas políticas são avaliadas para chegar a uma decisão de autorização, e o ponto de aplicação de políticas (PEP), que impõe essa decisão. O serviço de autorização hospedado, Permissões

verificadas, serve como PAP e PDP. Como alternativa, você mesmo pode criar seu PDP usando um mecanismo de código aberto, como Cedar ou OPA.

- Desacoplamento da lógica de autorização dos aplicativos — a lógica de autorização, quando incorporada ao código do aplicativo ou implementada por meio de um mecanismo de fiscalização ad hoc, pode estar sujeita a alterações acidentais ou maliciosas que causam acesso não intencional aos dados entre inquilinos ou outras violações de segurança. Para ajudar a mitigar essas possibilidades, você pode usar um PAP, como Permissões verificadas, para armazenar políticas de autorização independentemente do código do aplicativo e aplicar uma governança sólida ao gerenciamento dessas políticas. As políticas podem ser mantidas centralmente em uma linguagem declarativa de alto nível, o que torna a manutenção da lógica de autorização muito mais simples do que quando você incorpora políticas em várias seções do código do aplicativo. Essa abordagem também garante que as atualizações sejam aplicadas de forma consistente.
- Abordagem flexível para modelos de controle de acesso — Controle de acesso baseado em função (RBAC), controle de acesso baseado em atributos (ABAC) ou uma combinação dos dois modelos são todas abordagens válidas para controle de acesso. Esses modelos tentam atender aos requisitos de autorização de uma empresa usando abordagens diferentes. Este guia compara e contrasta esses modelos para ajudá-lo a selecionar um modelo que funcione para sua organização. O guia também discute como esses modelos se aplicam a diferentes linguagens de política de autorização, como OPA/Rego e Cedar. As arquiteturas discutidas neste guia permitem que um ou ambos os modelos sejam adotados com sucesso.
- Controle estrito de acesso à API — Este guia fornece um método para proteger de APIs forma consistente e abrangente um aplicativo com o mínimo esforço. Isso é particularmente valioso para arquiteturas de aplicativos orientados a serviços ou microsserviços que geralmente usam um grande número de APIs para facilitar as comunicações entre aplicativos. O controle rígido de acesso à API ajuda a aumentar a segurança de um aplicativo e o torna menos vulnerável a ataques ou exploração.

Isolamento de inquilinos e autorização de vários inquilinos

Este guia se refere aos conceitos de isolamento de inquilinos e autorização de vários inquilinos. O isolamento do inquilino se refere aos mecanismos explícitos que você usa em um sistema SaaS para garantir que os recursos de cada inquilino, mesmo quando operam em uma infraestrutura compartilhada, sejam isolados. A autorização multilocatária se refere a autorizar ações de entrada e impedir que elas sejam implementadas no inquilino errado. Um usuário hipotético poderia ser autenticado e autorizado e ainda acessar os recursos de outro inquilino. A autenticação e a

autorização não bloquearão esse acesso. Você precisa implementar o isolamento do inquilino para atingir esse objetivo. Para uma discussão mais ampla sobre as diferenças entre esses dois conceitos, consulte a seção Isolamento de inquilinos do whitepaper Fundamentos da Arquitetura SaaS.

Tipos de controle de acesso

Você pode usar dois modelos amplamente definidos para implementar o controle de acesso: controle de acesso baseado em função (RBAC) e controle de acesso baseado em atributos (ABAC). Cada modelo tem vantagens e desvantagens, que são discutidas brevemente nesta seção. O modelo que você deve usar depende do seu caso de uso específico. A arquitetura discutida neste guia é compatível com os dois modelos.

RBAC

O controle de acesso baseado em funções (RBAC) determina o acesso aos recursos com base em uma função que geralmente se alinha à lógica de negócios. As permissões são associadas à função, conforme apropriado. Por exemplo, uma função de marketing autorizaria um usuário a realizar atividades de marketing em um sistema restrito. Esse é um modelo de controle de acesso relativamente simples de implementar porque se alinha bem à lógica de negócios facilmente reconhecível.

O modelo RBAC é menos eficaz quando:

- Você tem usuários exclusivos cujas responsabilidades abrangem várias funções.
- Você tem uma lógica de negócios complexa que dificulta a definição de funções.
- A escalabilidade para um tamanho grande exige administração e mapeamento constantes de permissões para funções novas e existentes.
- As autorizações são baseadas em parâmetros dinâmicos.

ABAC

O controle de acesso baseado em atributos (ABAC) determina o acesso aos recursos com base nos atributos. Os atributos podem ser associados a um usuário, recurso, ambiente ou até mesmo ao estado do aplicativo. Suas políticas ou regras fazem referência a atributos e podem usar a lógica booleana básica para determinar se um usuário tem permissão para realizar uma ação. Aqui está um exemplo básico de permissões:

No sistema de pagamentos, todos os usuários do departamento financeiro podem processar pagamentos no endpoint da API `/payments` durante o horário comercial.

Ser membro do departamento financeiro é um atributo do usuário que determina o acesso `/payments` a. Também há um atributo de recurso associado ao endpoint da `/payments` API que permite acesso somente durante o horário comercial. No ABAC, se um usuário pode ou não processar um pagamento é determinado por uma política que inclui a associação ao departamento financeiro como atributo do usuário e a hora como atributo do recurso de`/payments`.

O modelo ABAC é muito flexível ao permitir decisões de autorização dinâmicas, contextuais e granulares. No entanto, o modelo ABAC é difícil de implementar inicialmente. Definir regras e políticas, bem como enumerar atributos para todos os vetores de acesso relevantes, exigem um investimento inicial significativo para serem implementadas.

Abordagem híbrida RBAC-ABAC

A combinação do RBAC e do ABAC pode oferecer algumas das vantagens de ambos os modelos. O RBAC, estando tão alinhado à lógica de negócios, é mais simples de implementar do que o ABAC. Para fornecer uma camada adicional de granularidade ao tomar decisões de autorização, você pode combinar o ABAC com o RBAC. Essa abordagem híbrida determina o acesso combinando a função de um usuário (e suas permissões atribuídas) com atributos adicionais para tomar decisões de acesso. O uso dos dois modelos permite uma administração e atribuição simples de permissões, além de permitir maior flexibilidade e granularidade em relação às decisões de autorização.

Comparação de modelos de controle de acesso

A tabela a seguir compara os três modelos de controle de acesso discutidos anteriormente. Essa comparação deve ser informativa e de alto nível. Usar um modelo de acesso em uma situação específica pode não necessariamente se correlacionar com as comparações feitas nesta tabela.

Fator	RBAC	ABAC	Híbrida
Flexibilidade	Médio	Alto	Alto
Simplicidade	Alto	Baixo	Médio
Granularity	Baixo	Alto	Médio
Decisões e regras dinâmicas	Não	Sim	Sim

Consciente do contexto	Não	Sim	Um pouco
Esforço de implementação	Baixo	Alto	Médio

Implementando um PDP

O ponto de decisão política (PDP) pode ser caracterizado como um mecanismo de políticas ou regras. Esse componente é responsável por aplicar políticas ou regras e retornar uma decisão sobre se um determinado acesso é permitido. Um PDP pode funcionar com modelos de controle de acesso baseado em função (RBAC) e controle de acesso baseado em atributos (ABAC); no entanto, um PDP é um requisito para o ABAC. Um PDP permite que a lógica de autorização no código do aplicativo seja transferida para um sistema separado. Isso pode simplificar o código do aplicativo. Ele também fornece uma interface easy-to-use repetível para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou qualquer outro componente do aplicativo.

As seções a seguir discutem três métodos para implementar um PDP. No entanto, essa não é uma lista completa.

Métodos de implementação do PDP:

- Implementando uma PDP usando as permissões verificadas da Amazon
- Implementando um PDP usando OPA
- Usando um mecanismo de política personalizado

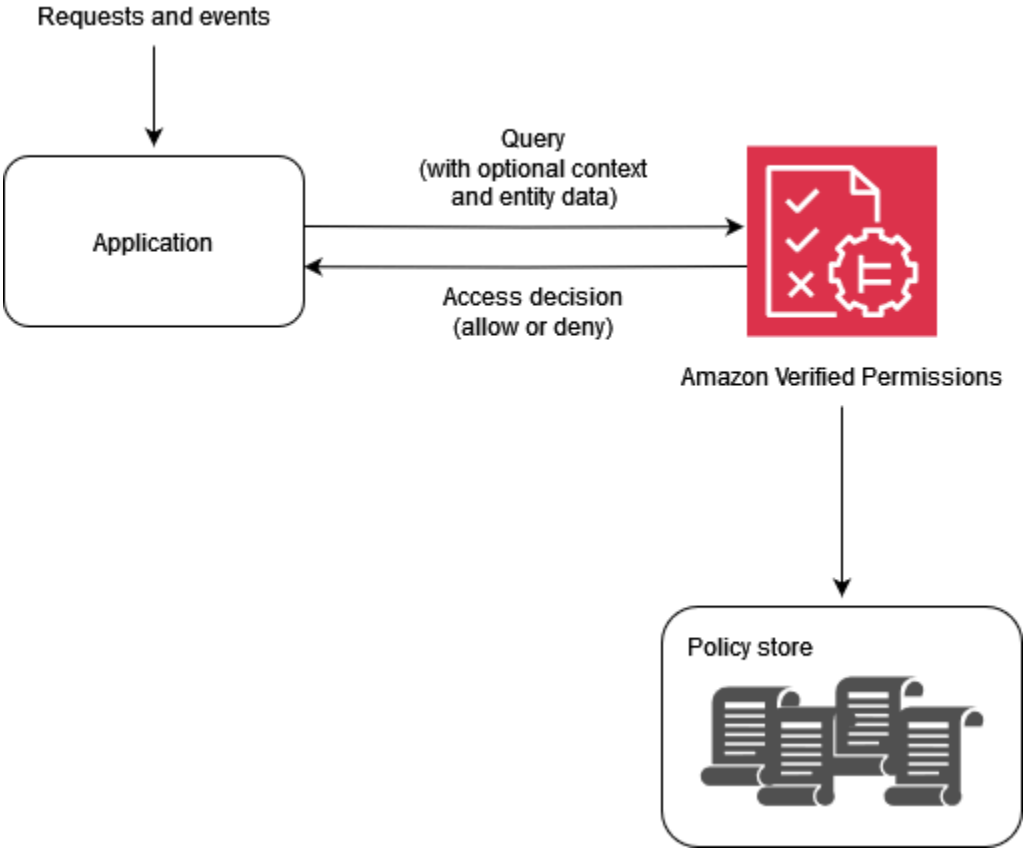
Implementando uma PDP usando as permissões verificadas da Amazon

O Amazon Verified Permissions é um serviço de autorização e gerenciamento de permissões escalável e refinado que você pode usar para implementar um ponto de decisão política (PDP). Como mecanismo de políticas, ele pode ajudar seu aplicativo a verificar as ações do usuário em tempo real e destacar permissões excessivamente privilegiadas ou inválidas. Ele ajuda seus desenvolvedores a criar aplicativos mais seguros com mais rapidez ao externalizar a autorização e centralizar o gerenciamento e a administração de políticas. Ao separar a lógica de autorização da lógica do aplicativo, as Permissões Verificadas oferecem suporte ao desacoplamento de políticas.

Ao usar Permissões Verificadas para implementar uma PDP e implementar privilégios mínimos e verificação contínua nos aplicativos, os desenvolvedores podem alinhar o acesso aos aplicativos com os princípios do Zero Trust. Além disso, as equipes de segurança e auditoria podem analisar e auditar melhor quem tem acesso a quais recursos em um aplicativo. O Verified Permissions usa

o Cedar, que é uma linguagem de política de código aberto criada especificamente e que prioriza a segurança, para definir controles de acesso baseados em políticas com base no controle de acesso baseado em funções (RBAC) e no controle de acesso baseado em atributos (ABAC) para um controle de acesso mais granular e sensível ao contexto.

As Permissões verificadas fornecem alguns recursos úteis para aplicativos SaaS, como a capacidade de habilitar a autorização de vários locatários usando vários provedores de identidade, como Amazon Cognito, Google e Facebook. Outro recurso de permissões verificadas que é particularmente útil para aplicativos SaaS é o suporte para funções personalizadas por inquilino. Se você estiver projetando um sistema de gerenciamento de relacionamento com o cliente (CRM), um inquilino pode definir a granularidade do acesso por oportunidades de vendas com base em um conjunto específico de critérios. Outro inquilino pode ter outra definição. Os sistemas de permissões subjacentes nas Permissões Verificadas podem suportar essas variações, o que o torna um excelente candidato para casos de uso de SaaS. As permissões verificadas também oferecem suporte à capacidade de criar políticas que se aplicam a todos os locatários, portanto, é fácil aplicar políticas de proteção para evitar o acesso não autorizado como provedor de SaaS.



Por que usar permissões verificadas?

Use Permissões verificadas com um provedor de identidade, como o Amazon Cognito, para obter uma solução de gerenciamento de acesso mais dinâmica e baseada em políticas para seus aplicativos. Você pode criar aplicativos que ajudem os usuários a compartilhar informações e colaborar, mantendo a segurança, a confidencialidade e a privacidade de seus dados. As permissões verificadas ajudam a reduzir os custos operacionais, fornecendo um sistema de autorização refinado para impor o acesso com base nas funções e atributos de suas identidades e recursos. Você pode definir seu modelo de política, criar e armazenar políticas em um local central e avaliar solicitações de acesso em milissegundos.

Em Permissões verificadas, você pode expressar permissões usando uma linguagem declarativa simples e legível por humanos chamada Cedar. As políticas escritas no Cedar podem ser compartilhadas entre as equipes, independentemente da linguagem de programação usada pelo aplicativo de cada equipe.

O que considerar ao usar permissões verificadas

Nas Permissões verificadas, você pode criar políticas e automatizá-las como parte do provisionamento. Você também pode criar políticas em tempo de execução como parte da lógica do aplicativo. Como prática recomendada, você deve usar um pipeline de integração contínua e implantação contínua (CI/CD) para administrar, modificar e rastrear versões de políticas ao criar políticas como parte da integração e provisionamento de inquilinos. Como alternativa, um aplicativo pode administrar, modificar e rastrear versões de políticas; no entanto, a lógica do aplicativo não executa inerentemente essa funcionalidade. Para oferecer suporte a esses recursos em seu aplicativo, você deve projetar explicitamente seu aplicativo para implementar essa funcionalidade.

Se for necessário fornecer dados externos de outras fontes para chegar a uma decisão de autorização, esses dados devem ser recuperados e fornecidos às Permissões Verificadas como parte da solicitação de autorização. Contexto, entidades e atributos adicionais não são recuperados por padrão com esse serviço.

Visão geral do cedro

O Cedar é uma linguagem de controle de acesso baseada em políticas flexível, extensível e escalável que ajuda os desenvolvedores a expressar permissões de aplicativos como políticas. Administradores e desenvolvedores podem definir políticas que permitam ou proíbam que os usuários atuem nos recursos do aplicativo. Várias políticas podem ser anexadas a um único recurso. Quando um usuário do seu aplicativo tenta realizar uma ação em um recurso, seu aplicativo solicita autorização do mecanismo de políticas do Cedar. A Cedar avalia as políticas aplicáveis e retorna

uma decisão `ALLOW` ou`DENY`. O Cedar suporta regras de autorização para qualquer tipo de principal e recurso, permite controle de acesso baseado em função (RBAC) e controle de acesso baseado em atributos (ABAC) e oferece suporte à análise por meio de ferramentas de raciocínio automatizadas.

O Cedar permite que você separe sua lógica de negócios da lógica de autorização. Ao fazer solicitações a partir do código do seu aplicativo, você chama o mecanismo de autorização do Cedar para determinar se a solicitação está autorizada. Se for autorizado (a decisão é`ALLOW`), seu aplicativo pode realizar a operação solicitada. Se não for autorizado (a decisão é`DENY`), seu aplicativo pode retornar uma mensagem de erro. As principais características do Cedar incluem:

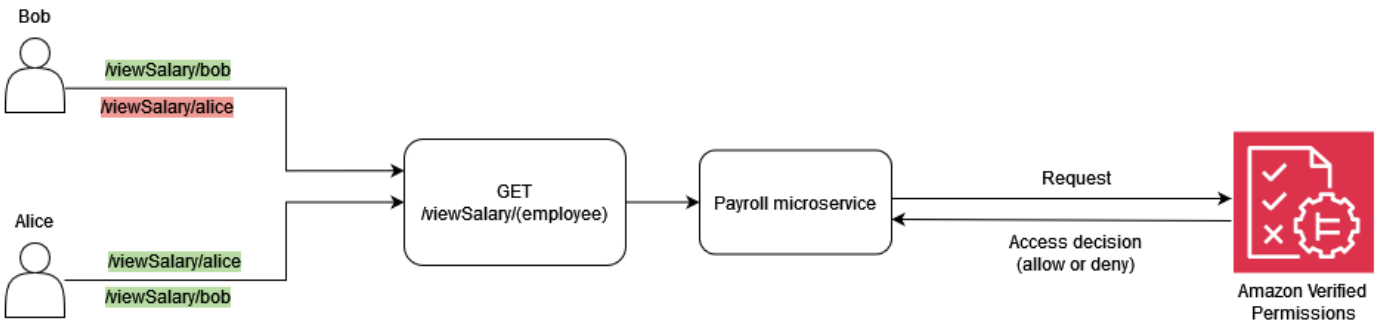
- Expressividade — O Cedar foi desenvolvido especificamente para apoiar casos de uso de autorização e foi desenvolvido pensando na legibilidade humana.
- Desempenho — O Cedar suporta políticas de indexação para recuperação rápida e fornece avaliação rápida e escalável em tempo real com latência limitada.
- Análise — O Cedar oferece suporte a ferramentas de análise que podem otimizar suas políticas e verificar seu modelo de segurança.

Para obter mais informações, consulte o site da Cedar.

Exemplo 1: ABAC básico com permissões verificadas e cedro

Neste cenário de exemplo, as Permissões Verificadas da Amazon são usadas para determinar quais usuários têm permissão para acessar informações em um microsserviço fictício de folha de pagamento. Esta seção inclui trechos de código do Cedar para demonstrar como você pode usar o Cedar para processar decisões de controle de acesso. Esses exemplos não pretendem fornecer uma exploração completa dos recursos fornecidos pelo Cedar e pelo Verified Permissions. Para uma visão geral mais completa do Cedar, consulte a documentação do Cedar.

No diagrama a seguir, gostaríamos de aplicar duas regras gerais de negócios associadas ao `viewSalary` `GET` método: os funcionários podem ver seu próprio salário e os funcionários podem ver o salário de qualquer pessoa que se reporte a eles. Você pode aplicar essas regras de negócios usando políticas de permissões verificadas.



Os funcionários podem ver seu próprio salário.

Em Cedar, a construção básica é uma entidade, que representa um principal, uma ação ou um recurso. Para fazer uma solicitação de autorização e iniciar uma avaliação com uma política de permissões verificadas, você precisa fornecer um principal, uma ação, um recurso e uma lista de entidades.

- O principal (`principal`) é o usuário ou função logado.
- A ação (`action`) é a operação que é avaliada pela solicitação.
- O recurso (`resource`) é o componente que a ação está acessando.
- A lista de entidades (`entityList`) contém todas as entidades necessárias para avaliar a solicitação.

Para satisfazer a regra de negócios, os funcionários podem ver seu próprio salário, você pode fornecer uma política de permissões verificadas, como a seguinte.

```
permit (
    principal,
    action == Action::"viewSalary",
    resource
)
when {
    principal == resource.owner
};
```

Essa política avalia `ALLOW` se o `Action` é `viewSalary` e o recurso na solicitação têm um proprietário de atributo igual ao principal. Por exemplo, se Bob for o usuário conectado que solicitou o relatório salarial e também for o proprietário do relatório salarial, a política será avaliada como. `ALLOW`

A solicitação de autorização a seguir é enviada às Permissões verificadas para ser avaliada pelo exemplo de política. Neste exemplo, Bob é o usuário logado que faz a `viewSalary` solicitação. Portanto, Bob é o principal do tipo de entidade`Employee`. A ação que Bob está tentando realizar é `viewSalary,` e o recurso que `viewSalary` será exibido é `Salary-Bob` com o tipo`Salary`. Para avaliar se Bob pode visualizar o `Salary-Bob` recurso, você precisa fornecer uma estrutura de entidade que vincule o tipo `Employee` com um valor de `Bob` (o principal) ao atributo proprietário do recurso que tem o tipo`Salary`. Você fornece essa estrutura em um`entityList`, em que os atributos associados `Salary` incluem um proprietário, que especifica um `entityIdentifier` que contém o tipo `Employee` e o valor`Bob`. As permissões verificadas comparam o `principal` fornecido na solicitação de autorização com o `owner` atributo associado ao `Salary` recurso para tomar uma decisão.

```
{
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Bob"
  },
  "action": {
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        }
```

```
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {}
      }
    ]
  }
}
```

A solicitação de autorização para Permissões verificadas retorna o seguinte como saída, em que o atributo `decision` é `ALLOW` ou`DENY`.

```
{
    "determiningPolicies":
        [
            {
                "determiningPolicyId": "PAYROLLAPP_POLICYSTOREID"
            }
        ],
    "decision": "ALLOW",
    "errors": []
}
```

Nesse caso, como Bob estava tentando ver seu próprio salário, a solicitação de autorização enviada para Permissões verificadas é avaliada como. `ALLOW` No entanto, nosso objetivo era usar as Permissões Verificadas para aplicar duas regras de negócios. A regra de negócios que afirma o seguinte também deve ser verdadeira:

Os funcionários podem ver o salário de qualquer pessoa que se reporte a eles.

Para satisfazer essa regra de negócios, você pode fornecer outra política. A política a seguir avalia `ALLOW` se a ação é `viewSalary` e se o recurso na solicitação tem um atributo `owner.manager` igual ao principal. Por exemplo, se Alice for a usuária conectada que solicitou o relatório salarial e Alice for a gerente do proprietário do relatório, a política será avaliada como. `ALLOW`

```
permit (
    principal,
    action == Action::"viewSalary",
```

```
    resource
)
when {
    principal == resource.owner.manager
};
```

A solicitação de autorização a seguir é enviada às Permissões verificadas para ser avaliada pelo exemplo de política. Neste exemplo, Alice é a usuária logada que faz a `viewSalary` solicitação. Portanto, Alice é a principal e a entidade é do tipo`Employee`. A ação que Alice está tentando realizar é`viewSalary`, e o recurso que `viewSalary` será exibido é do tipo `Salary` com um valor de`Salary-Bob`. Para avaliar se Alice pode visualizar o `Salary-Bob` recurso, você precisa fornecer uma estrutura de entidade que vincule o tipo `Employee` a um valor de `Alice` ao `manager` atributo, que deve então ser associado ao `owner` atributo do tipo `Salary` com um valor de`Salary-Bob`. Você fornece essa estrutura em um`entityList`, em que os atributos associados `Salary` incluem um proprietário, que especifica um `entityIdentifier` que contém o tipo `Employee` e o valor`Bob`. As permissões verificadas primeiro verificam o `owner` atributo, que avalia o tipo `Employee` e o valor`Bob`. Em seguida, as Permissões verificadas avaliam o `manager` atributo associado `Employee` e o comparam com o principal fornecido para tomar uma decisão de autorização. Nesse caso, a decisão é `ALLOW` porque os `resource.owner.manager` atributos `principal` e são equivalentes.

```
{
  "policyStoreId": "PAYROLLAPP_POLICYSTOREID",
  "principal": {
    "entityType": "PayrollApp::Employee",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "PayrollApp::Action",
    "actionId": "viewSalary"
  },
  "resource": {
    "entityType": "PayrollApp::Salary",
    "entityId": "Salary-Bob"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
```

```
          "entityId": "Alice"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "None"
            }
          }
        },
        "parents": []
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Salary",
          "entityId": "Salary-Bob"
        },
        "attributes": {
          "owner": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Bob"
            }
          }
        },
        "parents": []
      },
      {
        "identifier": {
          "entityType": "PayrollApp::Employee",
          "entityId": "Bob"
        },
        "attributes": {
          "manager": {
            "entityIdentifier": {
              "entityType": "PayrollApp::Employee",
              "entityId": "Alice"
            }
          }
        },
       "parents": []
      }
    ]
```

```
  }
}
```

Até agora, neste exemplo, fornecemos as duas regras de negócios associadas ao `viewSalary` método: os funcionários podem ver seu próprio salário e os funcionários podem ver o salário de qualquer pessoa que se reporte a eles. As permissões verificadas são políticas para satisfazer as condições de cada regra de negócios de forma independente. Você também pode usar uma única política de Permissões Verificadas para satisfazer as condições de ambas as regras de negócios:

Os funcionários podem ver seu próprio salário e o salário de qualquer pessoa que se reporte a eles.

Quando você usa a solicitação de autorização anterior, a política a seguir avalia `ALLOW` se a ação é `viewSalary` e se o recurso na solicitação tem um atributo `owner.manager` igual ao `principal` ou um atributo `owner` igual ao`principal`.

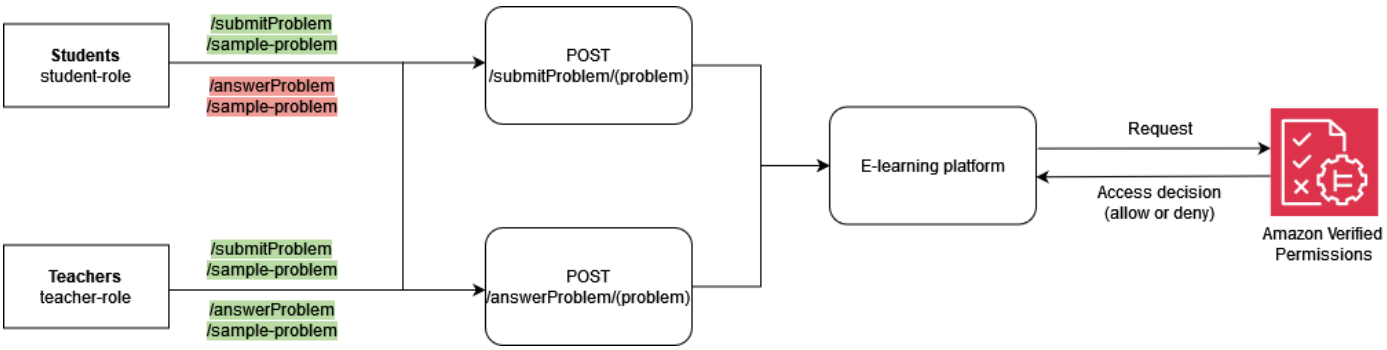
```
permit (
    principal,
    action == PayrollApp::Action::"viewSalary",
    resource
)
when {
    principal == resource.owner.manager ||
    principal == resource.owner
};
```

Por exemplo, se Alice for a usuária conectada que solicita o relatório salarial e se Alice for a gerente do proprietário ou a proprietária do relatório, a política será avaliada como. `ALLOW`

Para obter mais informações sobre o uso de operadores lógicos com políticas do Cedar, consulte a documentação do Cedar.

Exemplo 2: RBAC básico com permissões verificadas e Cedar

Este exemplo usa Permissões Verificadas e Cedar para demonstrar o RBAC básico. Conforme mencionado anteriormente, a construção básica do Cedar é uma entidade. Os desenvolvedores definem suas próprias entidades e, opcionalmente, podem criar relacionamentos entre entidades. O exemplo a seguir inclui três tipos de entidades: `UsersRoles`, `Problems` e. `Studentse Teachers` podem ser consideradas entidades do tipo `Role,` e cada uma `User` pode ser associada a zero ou a qualquer uma das`Roles`.



Em Cedar, essas relações são expressas vinculando o `Role Student` ao `User` Bob como seu pai. Essa associação agrupa logicamente todos os usuários estudantes em um grupo. Para obter mais informações sobre agrupamento no Cedar, consulte a documentação do Cedar.

A política a seguir avalia a decisão `ALLOW` da ação `submitProblem,` para todos os diretores vinculados ao grupo lógico `Students` do tipo. `Role`

```
permit (
    principal in ElearningApp::Role::"Students",
    action == ElearningApp::Action::"submitProblem",
    resource
);
```

A política a seguir avalia a decisão `ALLOW` da ação `submitProblem` ou`answerProblem`, de todos os diretores vinculados ao grupo lógico `Teachers` do tipo. `Role`

```
permit (
    principal in ElearningApp::Role::"Teachers",
    action in [
        ElearningApp::Action::"submitProblem",
        ElearningApp::Action::"answerProblem"
    ],
    resource
);
```

Para avaliar solicitações com essas políticas, o mecanismo de avaliação precisa saber se o principal referenciado na solicitação de autorização é membro do grupo apropriado. Portanto, o aplicativo deve passar informações relevantes de associação ao grupo para o mecanismo de avaliação como parte da solicitação de autorização. Isso é feito por meio da `entities` propriedade, que permite que você forneça ao mecanismo de avaliação do Cedar dados de atributo e associação ao grupo para

o principal e o recurso envolvido na chamada de autorização. No código a seguir, a associação ao grupo é indicada pela definição `User::"Bob"` de ter um pai chamado`Role::"Students"`.

```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Bob"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Bob"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
                    "entityId": "Students"
                }
            ]
        },
        {
          "identifier": {
            "entityType": "ElearningApp::Problem",
            "entityId": "SomeProblem"
          },
          "attributes": {},
          "parents": []
        }
     ]
  }
```

```
}
```

Neste exemplo, Bob é o usuário logado que faz a `answerProblem` solicitação. Portanto, Bob é o principal e a entidade é do tipo`User`. A ação que Bob está tentando realizar é`answerProblem`. Para avaliar se Bob pode realizar a `answerProblem` ação, você precisa fornecer uma estrutura de entidade que vincule a entidade `User` a um valor de `Bob` e atribua sua associação ao grupo listando uma entidade mãe como`Role::"Students"`. Como as entidades do grupo `Role::"Students"` de usuários só podem realizar a ação`submitProblem`, essa solicitação de autorização é avaliada como. `DENY`

Por outro lado, se o tipo `User` que tem um valor de `Alice` e faz parte do grupo `Role::"Teachers"` tentar realizar a `answerProblem` ação, a solicitação de autorização será avaliada`ALLOW`, pois a política determina que os diretores do grupo `Role::"Teachers"` tenham permissão para realizar a ação `answerProblem` em todos os recursos. O código a seguir mostra esse tipo de solicitação de autorização que avalia a. `ALLOW`

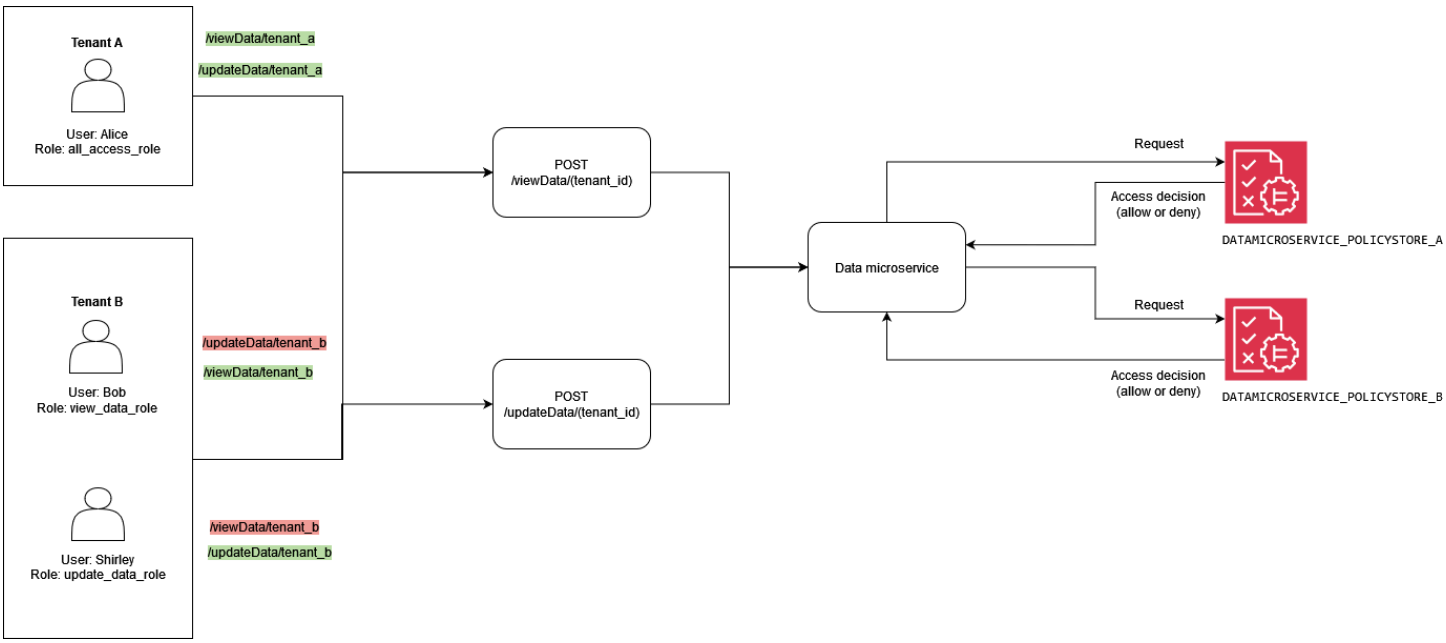
```
{
  "policyStoreId": "ELEARNING_POLICYSTOREID",
  "principal": {
    "entityType": "ElearningApp::User",
    "entityId": "Alice"
  },
  "action": {
    "actionType": "ElearningApp::Action",
    "actionId": "answerProblem"
  },
  "resource": {
    "entityType": "ElearningApp::Problem",
    "entityId": "SomeProblem"
  },
  "entities": {
    "entityList": [
        {
            "identifier": {
                "entityType": "ElearningApp::User",
                "entityId": "Alice"
            },
            "attributes": {},
            "parents": [
                {
                    "entityType": "ElearningApp::Role",
```

```
                    "entityId": "Teachers"
                }
            ]
        },
        {
            "identifier": {
                "entityType": "ElearningApp::Problem",
                "entityId": "SomeProblem"
            },
            "attributes": {},
            "parents": []
        }
    ]
  }
}
```

Exemplo 3: Controle de acesso multilocatário com RBAC

Para detalhar o exemplo anterior do RBAC, você pode expandir seus requisitos para incluir a multilocação de SaaS, que é um requisito comum para provedores de SaaS. Em soluções multilocatárias, o acesso aos recursos é sempre fornecido em nome de um determinado inquilino. Ou seja, os usuários do Locatário A não podem visualizar os dados do Locatário B, mesmo que esses dados estejam logicamente ou fisicamente colocados em um sistema. O exemplo a seguir ilustra como você pode implementar o isolamento de inquilinos usando vários repositórios de políticas de permissões verificadas e como você pode empregar funções de usuário para definir permissões dentro do inquilino.

Usar o padrão de design do Per Tenant Policy Store é uma prática recomendada para manter o isolamento do inquilino e, ao mesmo tempo, implementar o controle de acesso com permissões verificadas. Nesse cenário, as solicitações do usuário do Locatário A e do Locatário B são verificadas em relação a repositórios de políticas separados `DATAMICROSERVICE_POLICYSTORE_A` e`DATAMICROSERVICE_POLICYSTORE_B`, respectivamente. Para obter mais informações sobre as considerações de design de permissões verificadas para aplicativos SaaS multilocatários, consulte a seção Considerações sobre design de permissões verificadas para vários locatários.



A política a seguir reside no repositório de `DATAMICROSERVICE_POLICYSTORE_A` políticas. Ele verifica se o diretor fará parte do grupo `allAccessRole` do tipo`Role`. Nesse caso, o diretor poderá realizar as `updateData` ações `viewData` e em todos os recursos associados ao Locatário A.

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
    ],
    resource
);
```

As políticas a seguir residem no repositório `DATAMICROSERVICE_POLICYSTORE_B` de políticas. A primeira política verifica se o principal faz parte do `updateDataRole` grupo do tipo`Role`. Supondo que seja esse o caso, ele dá permissão aos diretores para realizar a `updateData` ação nos recursos associados ao Locatário B.

```
permit (
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
);
```

Essa segunda política determina que os diretores que fazem parte do `viewDataRole` grupo do tipo `Role` devem ter permissão para realizar a `viewData` ação em recursos associados ao Locatário B.

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
);
```

A solicitação de autorização feita pelo Locatário A precisa ser enviada ao repositório `DATAMICROSERVICE_POLICYSTORE_A` de políticas e verificada pelas políticas que pertencem a essa loja. Nesse caso, isso é verificado pela primeira política discutida anteriormente como parte deste exemplo. Nessa solicitação de autorização, o principal do tipo `User` com um valor de `Alice` está solicitando a execução da `viewData` ação. O principal pertence ao grupo `allAccessRole` do tipo`Role`. Alice está tentando realizar a `viewData` ação no `SampleData` recurso. Como Alice tem o `allAccessRole` papel, essa avaliação resulta em uma `ALLOW` decisão.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
            {
```

```
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

Se, em vez disso, você visualizar uma solicitação feita pelo Locatário B por`User Bob`, verá algo parecido com a seguinte solicitação de autorização. A solicitação é enviada para o repositório de `DATAMICROSERVICE_POLICYSTORE_B` políticas porque é originária do Locatário B. Nessa solicitação, o principal `Bob` deseja realizar a ação `updateData` no recurso. `SampleData` No entanto, não `Bob` faz parte de um grupo que tenha acesso à ação `updateData` nesse recurso. Portanto, a solicitação resulta em uma `DENY` decisão.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_B",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Bob"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
```

```
            "entityType": "MultitenantApp::User",
            "entityId": "Bob"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "viewDataRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

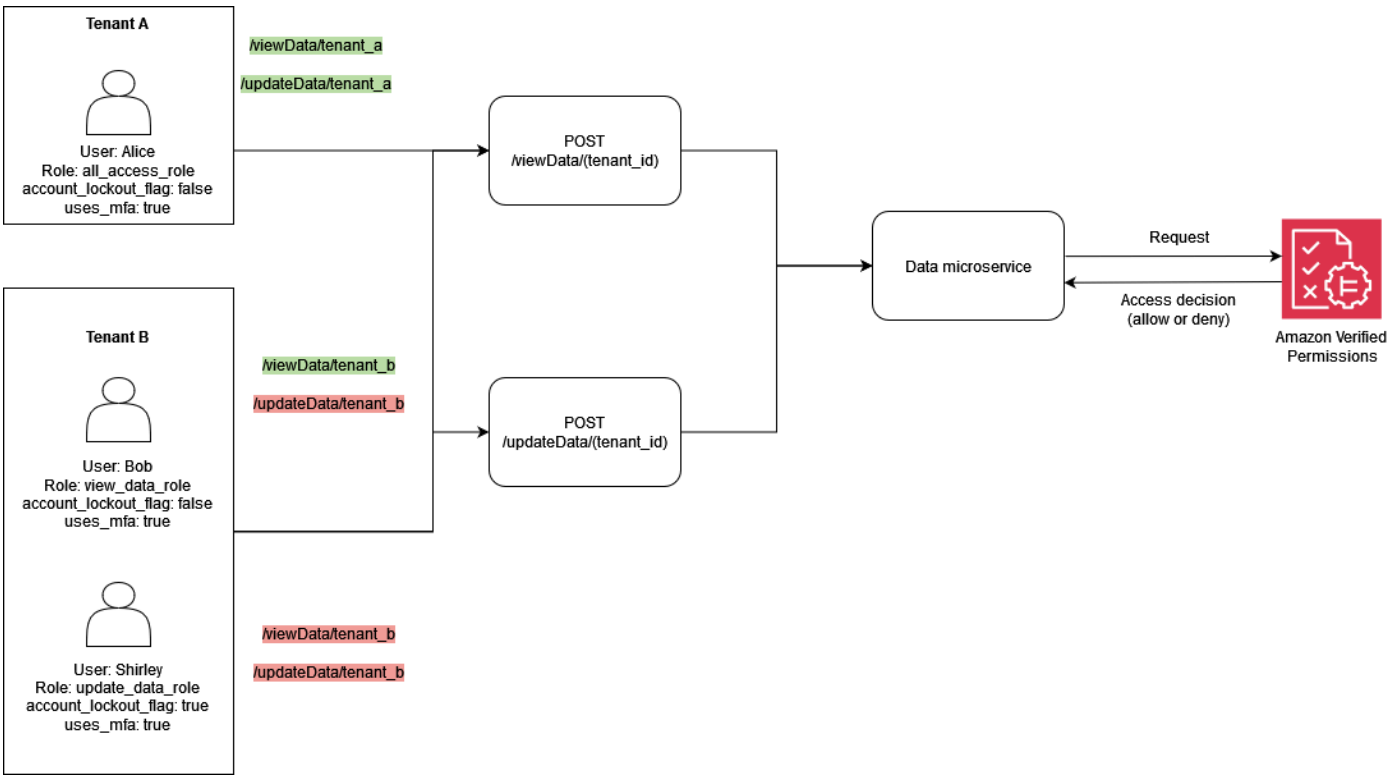
Neste terceiro exemplo, `User Alice` tenta realizar a `viewData` ação no recurso`SampleData`. Essa solicitação é direcionada ao repositório de `DATAMICROSERVICE_POLICYSTORE_A` políticas porque a diretora `Alice` pertence ao Locatário A. `Alice` faz parte do grupo `allAccessRole` do tipo`Role`, o que permite que ela execute a `viewData` ação nos recursos. Dessa forma, a solicitação resulta em uma `ALLOW` decisão.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE_A",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "viewData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
      "entityId": "SampleData"
```

```
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {},
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
      },
      {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": []
      }
    ]
  }
}
```

Exemplo 4: Controle de acesso multilocatário com RBAC e ABAC

Para aprimorar o exemplo de RBAC na seção anterior, você pode adicionar atributos aos usuários para criar uma abordagem híbrida RBAC-ABAC para controle de acesso multilocatário. Esse exemplo inclui as mesmas funções do exemplo anterior, mas adiciona o atributo de usuário `account_lockout_flag` e o parâmetro de contexto`uses_mfa`. O exemplo também adota uma abordagem diferente para implementar o controle de acesso multilocatário usando o RBAC e o ABAC, e usa um repositório de políticas compartilhado em vez de um armazenamento de políticas diferente para cada inquilino.



Este exemplo representa uma solução SaaS multilocatária na qual você precisa fornecer decisões de autorização para o Locatário A e o Locatário B, semelhante ao exemplo anterior.

Para implementar o recurso de bloqueio de usuário, o exemplo adiciona o `account_lockout_flag` atributo à `User` entidade principal na solicitação de autorização. Esse sinalizador bloqueia o acesso do usuário ao sistema e concede `DENY` todos os privilégios ao usuário bloqueado. O `account_lockout_flag` atributo está associado à `User` entidade e está em vigor `User` até que o sinalizador seja revogado ativamente em várias sessões. O exemplo usa a `when` condição para avaliar`account_lockout_flag`.

O exemplo também adiciona detalhes sobre a solicitação e a sessão. As informações de contexto especificam que a sessão foi autenticada usando a autenticação multifator. Para implementar essa validação, o exemplo usa a `when` condição para avaliar o `uses_mfa` sinalizador como parte do campo de contexto. Para obter mais informações sobre as melhores práticas para adicionar contexto, consulte a documentação do Cedar.

```
permit (
    principal in MultitenantApp::Role::"allAccessRole",
    action in [
        MultitenantApp::Action::"viewData",
        MultitenantApp::Action::"updateData"
```

```
    ],
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

Essa política impede o acesso aos recursos, a menos que o recurso esteja no mesmo grupo que o `Tenant` atributo do principal solicitante. Essa abordagem para manter o isolamento de inquilinos é conhecida como abordagem de um repositório de políticas compartilhado para vários locatários. Para obter mais informações sobre as considerações de design de permissões verificadas para aplicativos SaaS multilocatários, consulte a seção Considerações sobre design de permissões verificadas para vários locatários.

A política também garante que o diretor seja membro `allAccessRole` e restringe as ações de `viewData` e. `updateData` Além disso, essa política verifica se `account_lockout_flag` é `false` e se o valor do contexto de é `uses_mfa` avaliado como. `true`

Da mesma forma, a política a seguir garante que o principal e o recurso estejam associados ao mesmo inquilino, o que impede o acesso entre inquilinos. Essa política também garante que o diretor seja membro `viewDataRole` e restringe as ações a. `viewData` Além disso, verifica se o `account_lockout_flag` é `false` e se o valor do contexto para é `uses_mfa` avaliado como. `true`

```
permit (
    principal in MultitenantApp::Role::"viewDataRole",
    action == MultitenantApp::Action::"viewData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

A terceira política é semelhante à anterior. A política exige que o recurso seja membro do grupo que corresponde à entidade representada por`principal.Tenant`. Isso garante que tanto o principal quanto o recurso estejam associados ao Locatário B, o que impede o acesso entre inquilinos. Essa política garante que o diretor seja membro `updateDataRole` e restringe as ações a. `updateData`

Além disso, essa política verifica se o `account_lockout_flag` é `false` e se o valor do contexto para é `uses_mfa` avaliado como. `true`

```
permit (
    principal in MultitenantApp::Role::"updateDataRole",
    action == MultitenantApp::Action::"updateData",
    resource
)
when {
    principal.account_lockout_flag == false &&
    context.uses_mfa == true &&
    resource in principal.Tenant
};
```

A solicitação de autorização a seguir é avaliada pelas três políticas discutidas anteriormente nesta seção. Nessa solicitação de autorização, o principal do tipo `User` e com um valor de `Alice` faz uma `updateData` solicitação com a função`allAccessRole`. `Alice`tem o atributo `Tenant` cujo valor é`Tenant::"TenantA"`. A ação `Alice` que está tentando executar é `updateData,` e o recurso ao qual ela será aplicada é `SampleData` do tipo`Data`. `SampleData`tem `TenantA` como entidade controladora.

De acordo com a primeira <DATAMICROSERVICE_POLICYSTOREID> política no repositório de políticas, `Alice` pode executar a `updateData` ação no recurso, supondo que as condições na `when` cláusula da política sejam atendidas. A primeira condição exige que o `principal.Tenant` atributo seja avaliado`TenantA`. A segunda condição exige que o `account_lockout_flag` atributo do principal seja`false`. A condição final exige que `uses_mfa` o contexto seja`true`. Como todas as três condições foram atendidas, a solicitação retorna uma `ALLOW` decisão.

```
{
  "policyStoreId": "DATAMICROSERVICE_POLICYSTORE",
  "principal": {
      "entityType": "MultitenantApp::User",
      "entityId": "Alice"
  },
  "action": {
      "actionType": "MultitenantApp::Action",
      "actionId": "updateData"
  },
  "resource": {
      "entityType": "MultitenantApp::Data",
```

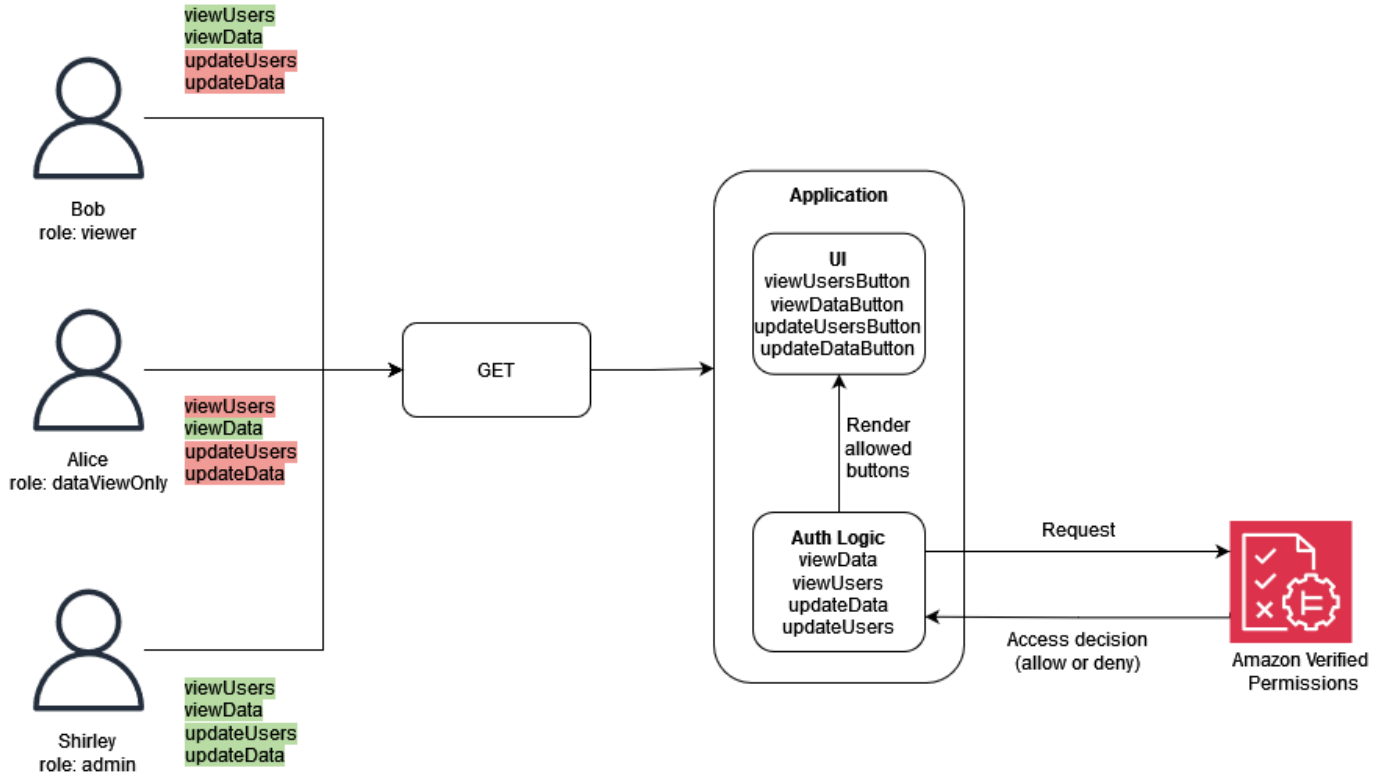
```
      "entityId": "SampleData"
  },
  "context": {
    "contextMap": {
        "uses_mfa": {
            "boolean": true
        }
    }
  },
  "entities": {
    "entityList": [
      {
        "identifier": {
            "entityType": "MultitenantApp::User",
            "entityId": "Alice"
        },
        "attributes": {
            {
                "account_lockout_flag": {
                    "boolean": false
                },
                "Tenant": {
                   "entityIdentifier": {
                        "entityType":"MultitenantApp::Tenant",
                        "entityId":"TenantA"
                   }
                }
            }
        },
        "parents": [
            {
                "entityType": "MultitenantApp::Role",
                "entityId": "allAccessRole"
            }
        ]
        },
     {
        "identifier": {
            "entityType": "MultitenantApp::Data",
            "entityId": "SampleData"
        },
        "attributes": {},
        "parents": [
```

```
            {
                "entityType": "MultitenantApp::Tenant",
                "entityId": "TenantA"
            }
        ]
      }
    ]
  }
}
```

Exemplo 5: filtragem de interface do usuário com permissões verificadas e Cedar

Você também pode usar as Permissões verificadas para implementar a filtragem RBAC dos elementos da interface do usuário com base em ações autorizadas. Isso é extremamente valioso para aplicativos que têm elementos de interface de usuário sensíveis ao contexto que podem estar associados a usuários ou inquilinos específicos no caso de um aplicativo SaaS multilocatário.

No exemplo a seguir, `Users` dos não `Role` `viewer` estão autorizados a realizar atualizações. Para esses usuários, a interface do usuário não deve renderizar nenhum botão de atualização.



Neste exemplo, um aplicativo web de página única tem quatro botões. Os botões visíveis dependem `Role` do usuário que está atualmente conectado ao aplicativo. À medida que o aplicativo web de página única renderiza a interface do usuário, ele consulta as permissões verificadas para determinar quais ações o usuário está autorizado a realizar e, em seguida, gera os botões com base na decisão de autorização.

A política a seguir especifica que o tipo `Role` com um valor de `viewer` pode visualizar usuários e dados. Uma decisão de `ALLOW` autorização para essa política exige uma `viewUsers` ação `viewData` or e também exige que um recurso seja associado ao tipo `Data` ou`Users`. Uma `ALLOW` decisão permite que a interface do usuário renderize dois botões: `viewDataButton` e. `viewUsersButton`

```
permit (
    principal in GuiAPP::Role::"viewer",
    action in [GuiAPP::Action::"viewData", GuiAPP::Action::"viewUsers"],
    resource
)
when {
   resource in [GuiAPP::Type::"Data", GuiAPP::Type::"Users"]
};
```

A política a seguir especifica que o tipo `Role` com um valor de só `viewerDataOnly` pode visualizar dados. Uma decisão de `ALLOW` autorização para essa política exige uma `viewData` ação e também exige que um recurso seja associado ao tipo`Data`. Uma `ALLOW` decisão permite que a interface do usuário renderize o botão`viewDataButton`.

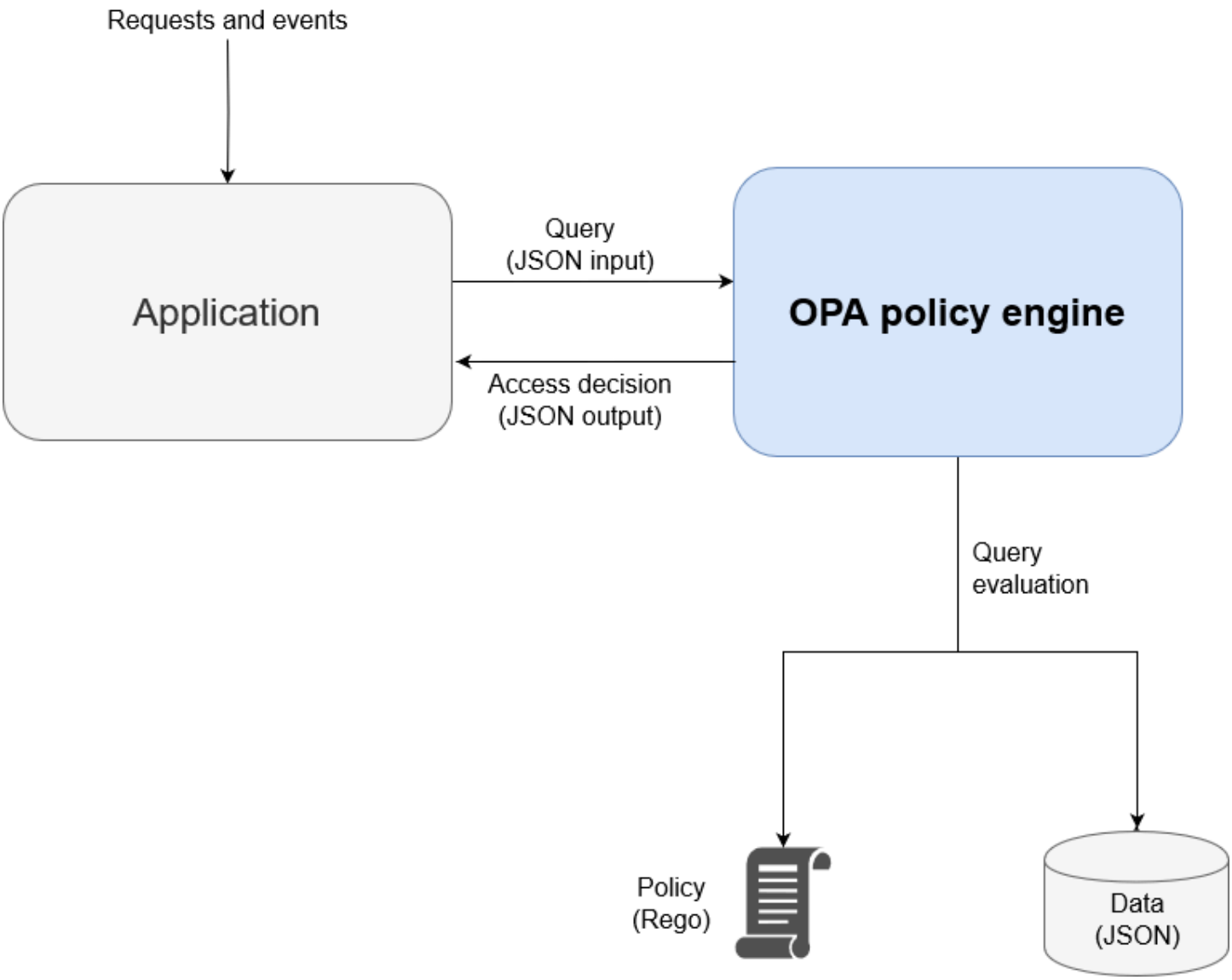
```
permit (
    principal in GuiApp::Role::"viewerDataOnly",
    action in [GuiApp::Action::"viewData"],
    resource in [GuiApp::Type::"Data"]
);
```

A política a seguir especifica que o tipo `Role` com um valor de `admin` pode editar e visualizar dados e usuários. Uma decisão de `ALLOW` autorização para essa política exige uma ação de `updateDataupdateUsers`, `viewData`, ou`viewUsers`, e também exige que um recurso seja associado ao tipo `Data` ou`Users`. Uma `ALLOW` decisão permite que a interface do usuário renderize todos os quatro botões: `updateDataButton`,`updateUsersButton`, `viewDataButton`, e. `viewUsersButton`

```
permit (
    principal in GuiApp::Role::"admin",
    action in [
        GuiApp::Action::"updateData",
        GuiApp::Action::"updateUsers",
        GuiApp::Action::"viewData",
        GuiApp::Action::"viewUsers"
       ],
    resource
)
when {
   resource in [GuiApp::Type::"Data", GuiApp::Type::"Users"]
};
```

Implementando um PDP usando OPA

O Open Policy Agent (OPA) é um mecanismo de políticas de uso geral de código aberto. O OPA tem muitos casos de uso, mas o caso de uso relevante para a implementação do PDP é sua capacidade de desacoplar a lógica de autorização de um aplicativo. Isso é chamado de desacoplamento de políticas. O OPA é útil na implementação de um PDP por vários motivos. Ele usa uma linguagem declarativa de alto nível chamada Rego para elaborar políticas e regras. Essas políticas e regras existem separadamente de um aplicativo e podem processar decisões de autorização sem nenhuma lógica específica do aplicativo. O OPA também expõe uma RESTful API para tornar as decisões de autorização de recuperação simples e diretas. Para tomar uma decisão de autorização, um aplicativo consulta a OPA com entrada JSON e a OPA avalia a entrada em relação às políticas especificadas para retornar uma decisão de acesso em JSON. O OPA também é capaz de importar dados externos que podem ser relevantes para a tomada de uma decisão de autorização.



O OPA tem várias vantagens em relação aos mecanismos de políticas personalizadas:

- A OPA e sua avaliação de políticas com a Rego fornecem um mecanismo de políticas flexível e pré-construído que requer apenas a inserção de políticas e quaisquer dados necessários para tomar decisões de autorização. Essa lógica de avaliação de políticas precisaria ser recriada em uma solução personalizada de mecanismo de políticas.
- O OPA simplifica a lógica de autorização ao ter políticas escritas em uma linguagem declarativa. Você pode modificar e administrar essas políticas e regras independentemente de qualquer código de aplicativo, sem habilidades de desenvolvimento de aplicativos.
- O OPA expõe uma RESTful API, que simplifica a integração com pontos de aplicação de políticas (). PEPs

- O OPA fornece suporte integrado para validar e decodificar JSON Web Tokens (). JWTs
- O OPA é um padrão de autorização reconhecido, o que significa que a documentação e os exemplos são abundantes se você precisar de ajuda ou pesquisa para resolver um problema específico.
- A adoção de um padrão de autorização, como o OPA, permite que as políticas escritas em Rego sejam compartilhadas entre as equipes, independentemente da linguagem de programação usada pelo aplicativo da equipe.

Há duas coisas que a OPA não fornece automaticamente:

- A OPA não tem um plano de controle robusto para atualizar e gerenciar políticas. O OPA fornece alguns padrões básicos para implementar atualizações de políticas, monitoramento e agregação de registros ao expor uma API de gerenciamento, mas a integração com essa API deve ser feita pelo usuário do OPA. Como prática recomendada, você deve usar um pipeline de integração contínua e implantação contínua (CI/CD) para administrar, modificar e rastrear versões de políticas e gerenciar políticas no OPA.
- O OPA não pode recuperar dados de fontes externas por padrão. Uma fonte externa de dados para uma decisão de autorização pode ser um banco de dados que contém atributos do usuário. Há alguma flexibilidade na forma como os dados externos são fornecidos ao OPA — eles podem ser armazenados em cache localmente com antecedência ou recuperados dinamicamente de uma API quando uma decisão de autorização é solicitada — mas obter essas informações não é algo que o OPA possa fazer em seu nome.

Visão geral de Rego

O Rego é uma linguagem de política de uso geral, o que significa que funciona para qualquer camada da pilha e qualquer domínio. O objetivo principal do Rego é aceitar entradas e dados JSON/YAML que são avaliados para tomar decisões baseadas em políticas sobre recursos, identidades e operações de infraestrutura. O Rego permite que você escreva políticas sobre qualquer camada de uma pilha ou domínio sem exigir uma alteração ou extensão da linguagem. Aqui estão alguns exemplos de decisões que Rego pode tomar:

- Essa solicitação de API é permitida ou negada?
- Qual é o nome do host do servidor de backup desse aplicativo?
- Qual é a pontuação de risco dessa mudança de infraestrutura proposta?

- Em quais clusters esse contêiner deve ser implantado para obter alta disponibilidade?
- Quais informações de roteamento devem ser usadas para esse microsserviço?

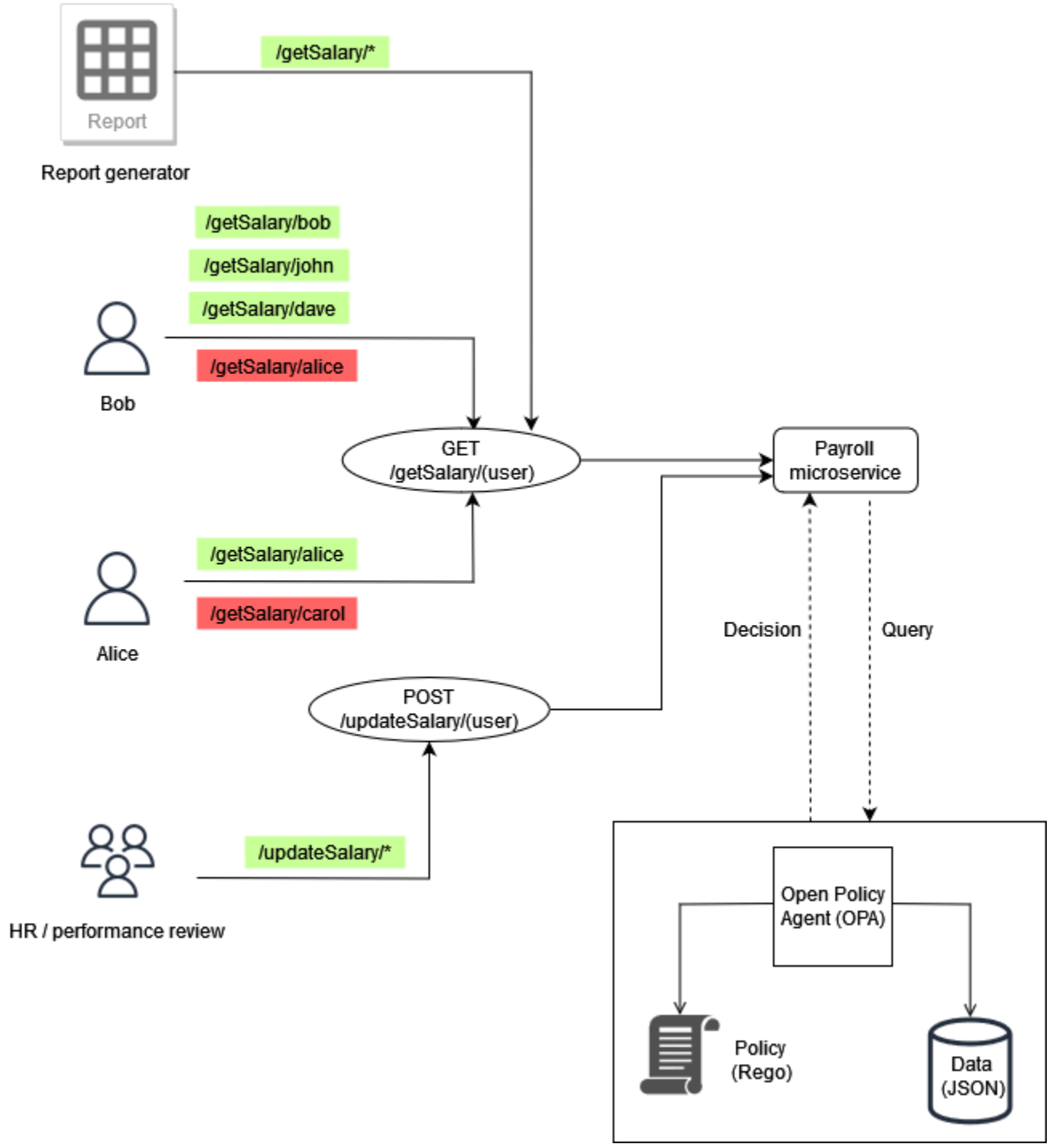
Para responder a essas perguntas, Rego emprega uma filosofia básica sobre como essas decisões podem ser tomadas. Os dois princípios fundamentais na elaboração de políticas em Rego são:

- Cada recurso, identidade ou operação pode ser representado como dados JSON ou YAML.
- A política é a lógica aplicada aos dados.

O Rego ajuda os sistemas de software a tomar decisões de autorização definindo a lógica sobre como as entradas de JSON/YAML dados são avaliadas. Linguagens de programação como C, Java, Go e Python são a solução usual para esse problema, mas o Rego foi projetado para se concentrar nos dados e entradas que representam seu sistema e na lógica para tomar decisões políticas com essas informações.

Exemplo 1: ABAC básico com OPA e Rego

Esta seção descreve um cenário em que o OPA é usado para tomar decisões de acesso sobre quais usuários têm permissão para acessar informações em um microsserviço fictício de folha de pagamento. Trechos de código do Rego são fornecidos para demonstrar como você pode usar o Rego para renderizar decisões de controle de acesso. Esses exemplos não são exaustivos nem uma exploração completa das capacidades do Rego e do OPA. Para uma visão geral mais completa do Rego, recomendamos que você consulte a documentação do Rego no site da OPA.



Exemplo de regras básicas de OPA

No diagrama anterior, uma das regras de controle de acesso aplicadas pela OPA para o microsserviço de folha de pagamento é:

Os funcionários podem ler seu próprio salário.

Se Bob tentar acessar o microsserviço da folha de pagamento para ver seu próprio salário, o microsserviço da folha de pagamento poderá redirecionar a chamada da API para a API da OPA para tomar uma RESTful decisão de acesso. O serviço de folha de pagamento consulta a OPA para obter uma decisão com a seguinte entrada JSON:

```
{
    "user": "bob",
    "method": "GET",
    "path": ["getSalary", "bob"]
}
```

O OPA seleciona uma política ou políticas com base na consulta. Nesse caso, a política a seguir, escrita em Rego, avalia a entrada JSON.

```
default allow = false
allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}
```

Essa política nega acesso por padrão. Em seguida, ele avalia a entrada na consulta vinculando-a à variável `input` global. O operador ponto é usado com essa variável para acessar os valores da variável. A regra Rego `allow` retorna verdadeira se as expressões na regra também forem verdadeiras. A regra Rego verifica se a `method` entrada é igual a GET. Em seguida, ele verifica se o primeiro elemento na lista `path` está `getSalary` antes de atribuir o segundo elemento na lista à variável. `user` Por fim, ele verifica se o caminho que está sendo `/getSalary/bob` acessado é verificando se a `user` solicitação,`input.user`, corresponde à `user` variável. A regra `allow` aplica a lógica if-then para retornar um valor booleano, conforme mostrado na saída:

```
{
    "allow": true
}
```

Regra parcial usando dados externos

Para demonstrar recursos adicionais do OPA, você pode adicionar requisitos à regra de acesso que está aplicando. Vamos supor que você queira impor esse requisito de controle de acesso no contexto da ilustração anterior:

Os funcionários podem ler o salário de qualquer pessoa que se reporte a eles.

Neste exemplo, o OPA tem acesso a dados externos que podem ser importados para ajudar a tomar uma decisão de acesso:

```
"managers": {
        "bob": ["dave", "john"],
        "carol": ["alice"]
}
```

Você pode gerar uma resposta JSON arbitrária criando uma regra parcial no OPA, que retorna um conjunto de valores em vez de uma resposta fixa. Este é um exemplo de uma regra parcial:

```
direct_report[user_ids] {
    user_ids = data.managers[input.user][_]
}
```

Essa regra retorna um conjunto de todos os usuários que se reportam ao valor de`input.user`, que, nesse caso, é`bob`. A `[_]` construção na regra é usada para iterar sobre os valores do conjunto. Esta é a saída da regra:

```
{
    "direct_report": [
      "dave",
      "john"
    ]
}
```

A recuperação dessas informações pode ajudar a determinar se um usuário é subordinado direto de um gerente. Para alguns aplicativos, é preferível retornar JSON dinâmico do que retornar uma resposta booleana simples.

Juntando tudo isso

O último requisito de acesso é mais complexo do que os dois primeiros porque combina as condições especificadas nos dois requisitos:

Os funcionários podem ler seu próprio salário e o salário de qualquer pessoa que se reporte a eles.

Para cumprir esse requisito, você pode usar esta política da Rego:

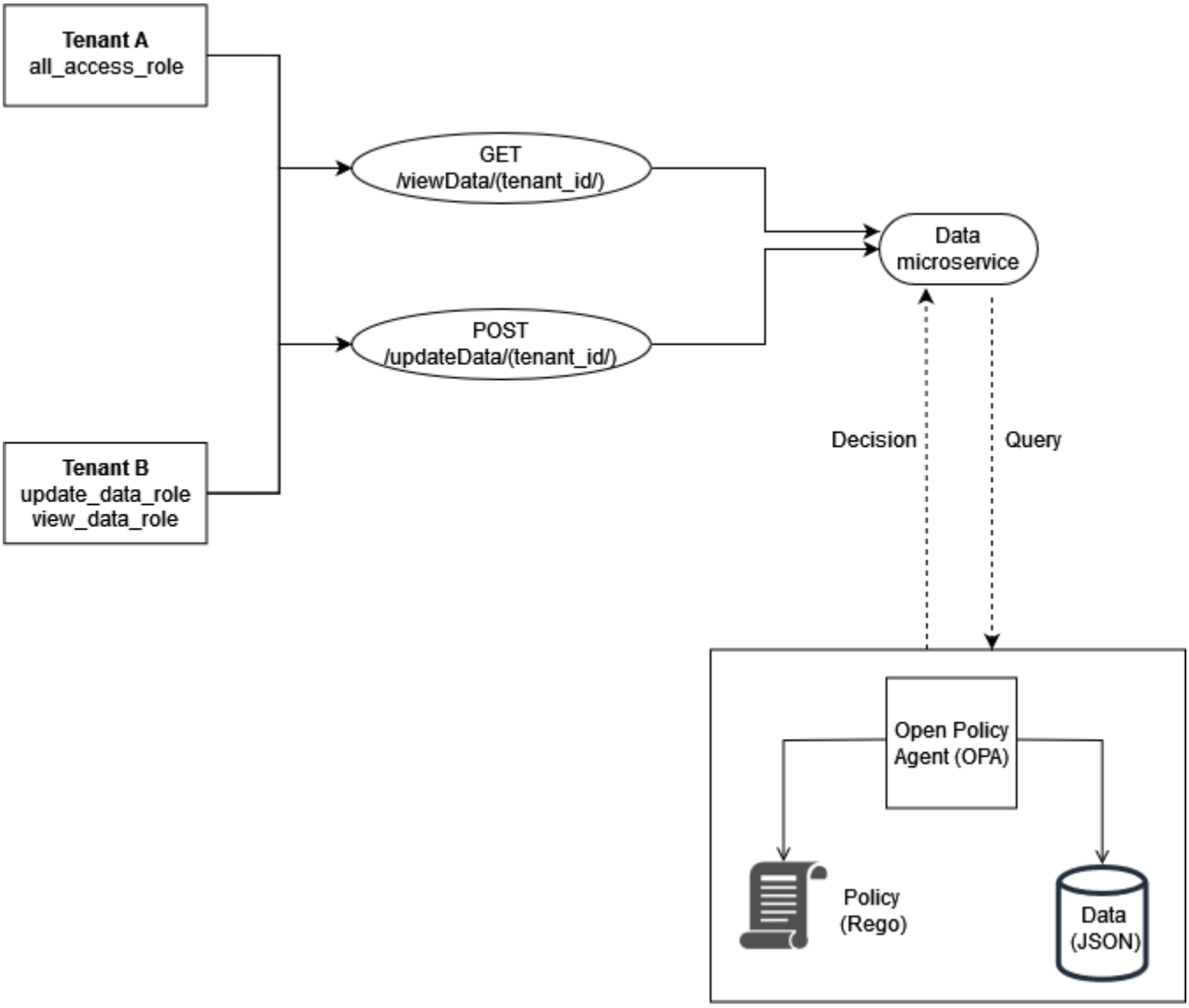
```
default allow = false

allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    input.user == user
}

allow = true {
    input.method == "GET"
    input.path = ["getSalary", user]
    managers := data.managers[input.user][_]
    contains(managers, user)
}
```

A primeira regra da política permite o acesso de qualquer usuário que tente ver suas próprias informações salariais, conforme discutido anteriormente. Ter duas regras com o mesmo nome,`allow`, funciona como um operador lógico ou em Rego. A segunda regra recupera a lista de todos os subordinados diretos associados `input.user` (dos dados no diagrama anterior) e atribui essa lista à `managers` variável. Por fim, a regra verifica se o usuário que está tentando ver seu salário é um subordinado direto, `input.user` verificando se seu nome está contido na `managers` variável.

Os exemplos nesta seção são muito básicos e não fornecem uma exploração completa ou completa das capacidades do Rego e do OPA. Para obter mais informações, revise a documentação do OPA, consulte o arquivo GitHub README do OPA e experimente no playground do Rego.

Exemplo 2: Controle de acesso multilocatário e RBAC definido pelo usuário com OPA e Rego

Este exemplo usa OPA e Rego para demonstrar como o controle de acesso pode ser implementado em uma API para um aplicativo multilocatário com funções personalizadas definidas pelos usuários locatários. Também demonstra como o acesso pode ser restrito com base em um inquilino.
Esse modelo mostra como a OPA pode tomar decisões granulares de permissão com base nas informações fornecidas em uma função de alto nível.



As funções dos inquilinos são armazenadas em dados externos (dados RBAC) que são usados para tomar decisões de acesso ao OPA:

```
{
    "roles": {
        "tenant_a": {
            "all_access_role": ["viewData", "updateData"]
        },
        "tenant_b": {
            "update_data_role": ["updateData"],
            "view_data_role": ["viewData"]
        }
    }
```

```
}
```

Essas funções, quando definidas por um usuário inquilino, devem ser armazenadas em uma fonte de dados externa ou em um provedor de identidade (IdP) que possa atuar como uma fonte confiável ao mapear funções definidas pelo inquilino para as permissões e para o próprio inquilino.

Este exemplo usa duas políticas no OPA para tomar decisões de autorização e examinar como essas políticas impõem o isolamento do inquilino. Essas políticas usam os dados RBAC definidos anteriormente.

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
}
```

Para mostrar como essa regra funcionará, considere uma consulta OPA que tenha a seguinte entrada:

```
{
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET"
}
```

Uma decisão de autorização para essa chamada de API é feita da seguinte forma, combinando os dados do RBAC, as políticas de OPA e a entrada da consulta OPA:

1. Um usuário de `Tenant A` faz uma chamada de API para`/viewData/tenant_a`.
2. O microsserviço Data recebe a chamada e consulta a `allowViewData` regra, passando a entrada mostrada no exemplo de entrada da consulta OPA.
3. A OPA usa a regra consultada nas políticas de OPA para avaliar a entrada fornecida. O OPA também usa os dados dos dados do RBAC para avaliar a entrada. A OPA faz o seguinte:
   a. Verifica se o método usado para fazer a chamada da API é`GET`.
   b. Verifica se o caminho solicitado é`viewData`.

c. Verifica se o `tenant_id` no caminho é igual ao `input.tenant_id` associado ao usuário. Isso garante que o isolamento do inquilino seja mantido. Outro inquilino, mesmo com uma função idêntica, não pode ser autorizado a fazer essa chamada de API.
d. Extrai uma lista de permissões de função dos dados externos das funções e as atribui à variável. `role_permissions` Essa lista é recuperada usando a função definida pelo inquilino associada ao usuário no `input.role.`
e. Verifica `role_permissions` se ele contém a permissão `viewData.`

4. A OPA retorna a seguinte decisão ao microsserviço Data:

```
{
    "allowViewData": true
}
```

Esse processo mostra como o RBAC e a conscientização do inquilino podem contribuir para a tomada de uma decisão de autorização com a OPA. Para ilustrar melhor esse ponto, considere uma chamada de API para `/viewData/tenant_b` com a seguinte entrada de consulta:

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["viewData", "tenant_b"],
    "method": "GET"
}
```

Essa regra retornaria a mesma saída da entrada da consulta OPA, embora seja para um inquilino diferente que tenha uma função diferente. Isso ocorre porque essa chamada é para `/tenant_b` e os `view_data_role` dados do RBAC ainda têm a `viewData` permissão associada a ela. Para aplicar o mesmo tipo de controle de acesso`/updateData`, você pode usar uma regra de OPA semelhante:

```
default allowUpdateData = false
allowUpdateData = true {
    input.method == "POST"
    input.path = ["updateData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "updateData")
}
```

Essa regra é funcionalmente igual à `allowViewData` regra, mas verifica um caminho e um método de entrada diferentes. A regra ainda garante o isolamento do inquilino e verifica se a função definida pelo inquilino concede permissão ao chamador da API. Para ver como isso pode ser aplicado, examine a seguinte entrada de consulta para uma chamada de API para`/updateData/tenant_b`:

```
{
    "tenant_id": "tenant_b",
    "role": "view_data_role",
    "path": ["updateData", "tenant_b"],
    "method": "POST"
}
```

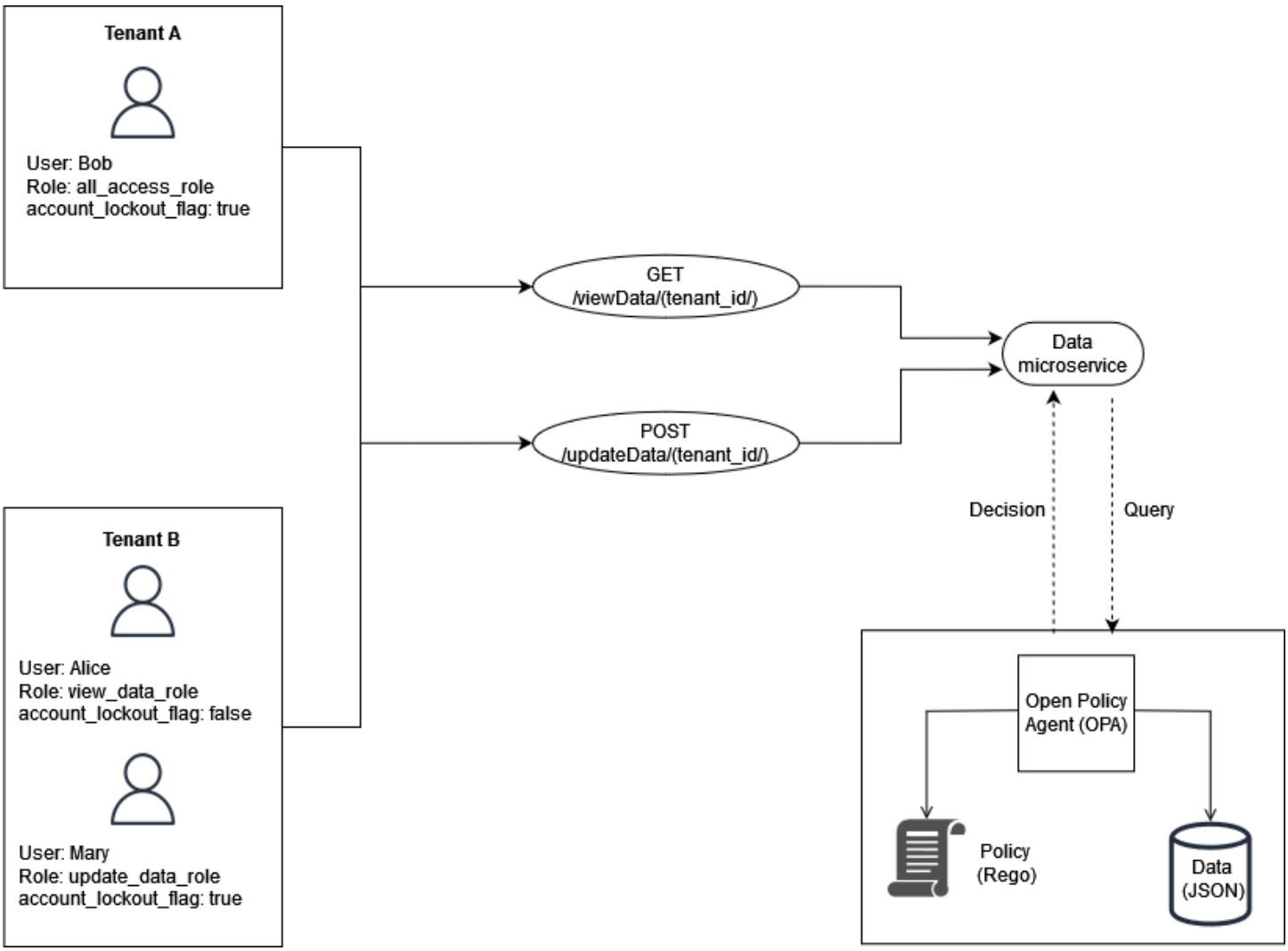
Essa entrada de consulta, quando avaliada com a `allowUpdateData` regra, retorna a seguinte decisão de autorização:

```
{
    "allowUpdateData": false
}
```

Essa chamada não será autorizada. Embora o chamador da API esteja associado ao correto `tenant_id` e esteja chamando a API usando um método aprovado, ele `input.role` é definido pelo inquilino`view_data_role`. `view_data_role`Não tem `updateData` permissão; portanto, a chamada para não `/updateData` é autorizada. Essa chamada teria sido bem-sucedida para um `tenant_b` usuário que tem `update_data_role` o.

Exemplo 3: Controle de acesso multilocatário para RBAC e ABAC com OPA e Rego

Para aprimorar o exemplo do RBAC na seção anterior, você pode adicionar atributos aos usuários.



Esse exemplo inclui as mesmas funções do exemplo anterior, mas adiciona o atributo de usuário`account_lockout_flag`. Esse é um atributo específico do usuário que não está associado a nenhuma função específica. Você pode usar os mesmos dados externos do RBAC usados anteriormente neste exemplo:

```
{
   "roles": {
       "tenant_a": {
           "all_access_role": ["viewData", "updateData"]
       },
       "tenant_b": {
           "update_data_role": ["updateData"],
           "view_data_role": ["viewData"]
       }
   }
```

```
}
```

O atributo `account_lockout_flag` do usuário pode ser passado para o serviço de dados como parte da entrada de uma consulta OPA `/viewData/tenant_a` para o usuário Bob:

```
 {
    "tenant_id": "tenant_a",
    "role": "all_access_role",
    "path": ["viewData", "tenant_a"],
    "method": "GET",
    "account_lockout_flag": "true"
}
```

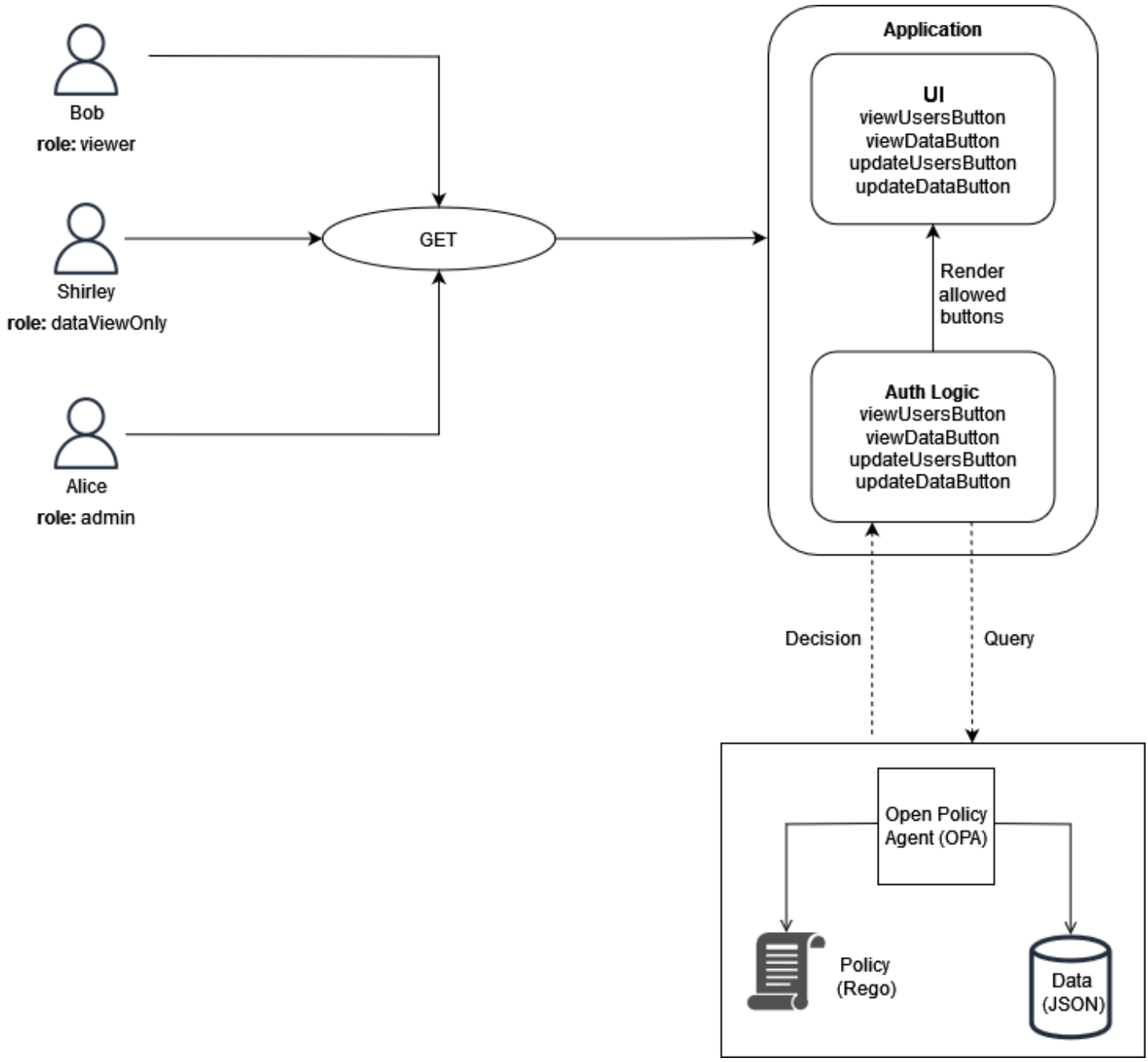
A regra que é consultada para a decisão de acesso é semelhante aos exemplos anteriores, mas inclui uma linha adicional para verificar o `account_lockout_flag` atributo:

```
default allowViewData = false
allowViewData = true {
    input.method == "GET"
    input.path = ["viewData", tenant_id]
    input.tenant_id == tenant_id
    role_permissions := data.roles[input.tenant_id][input.role][_]
    contains(role_permissions, "viewData")
    input.account_lockout_flag == "false"
}
```

Essa consulta retorna uma decisão de autorização de`false`. Isso ocorre porque `account_lockout_flag attribute` é `true` para Bob, e a regra Rego `allowViewData` nega o acesso, embora Bob tenha a função e o inquilino corretos.

Exemplo 4: filtragem de interface do usuário com OPA e Rego

A flexibilidade do OPA e do Rego suporta a capacidade de filtrar elementos da interface do usuário. O exemplo a seguir demonstra como uma regra parcial de OPA pode tomar decisões de autorização sobre quais elementos devem ser exibidos em uma interface de usuário com RBAC. Esse método é uma das muitas maneiras diferentes de filtrar elementos da interface do usuário com o OPA.



Neste exemplo, um aplicativo web de página única tem quatro botões. Digamos que você queira filtrar a interface de usuário de Bob, Shirley e Alice para que eles possam ver somente os botões que correspondem às suas funções. Quando a interface recebe uma solicitação do usuário, ela consulta uma regra parcial do OPA para determinar quais botões devem ser exibidos na interface do usuário. A consulta passa o seguinte como entrada para a OPA quando Bob (com a função`viewer`) faz uma solicitação para a interface do usuário:

```
{
```

```
    "role": "viewer"
}
```

A OPA usa dados externos estruturados para o RBAC para tomar uma decisão de acesso:

```
{
    "roles": {
        "viewer": ["viewUsers", "viewData"],
        "dataViewOnly": ["viewData"],
        "admin": ["viewUsers", "viewData", "updateUsers", "updateData"]
    }
}
```

A regra parcial do OPA usa os dados externos e a entrada para produzir uma lista de ações permitidas:

```
user_permissions[permissions] {
    permissions := data.roles[input.role][_]
}
```

Na regra parcial, o OPA usa o `input.role` especificado como parte da consulta para determinar quais botões devem ser exibidos. Bob tem a funçãoviewer, e os dados externos especificam que os espectadores têm duas permissões: `viewUsers` e. `viewData` Portanto, a saída dessa regra para Bob (e para qualquer outro usuário que tenha uma função de visualizador) é a seguinte:

```
{
    "user_permissions": [
        "viewData",
        "viewUsers"
    ]
}
```

A saída para Shirley, quem tem a `dataViewOnly` função, conteria um botão de permissões:. `viewData` A saída para Alice, que tem a `admin` função, conteria todas essas permissões. Essas respostas são retornadas à interface do usuário quando a OPA é consultada. `user_permissions` O aplicativo pode então usar essa resposta para ocultar ou exibir o `viewUsersButton` `viewDataButton`updateUsersButton,, `updateDataButton` e.

Usando um mecanismo de política personalizado

Um método alternativo para implementar um PDP é criar um mecanismo de política personalizado. O objetivo desse mecanismo de política é desacoplar a lógica de autorização de um aplicativo. O mecanismo de política personalizada é responsável por tomar decisões de autorização, semelhantes às Permissões Verificadas ou OPA, para obter a dissociação de políticas. A principal diferença entre essa solução e o uso de Permissões Verificadas ou OPA é que a lógica para escrever e avaliar políticas é personalizada para um mecanismo de políticas personalizado. Qualquer interação com o mecanismo deve ser exposta por meio de uma API ou algum outro método para permitir que as decisões de autorização cheguem a um aplicativo. Você pode escrever um mecanismo de política personalizado em qualquer linguagem de programação ou usar outros mecanismos para avaliação de políticas, como a Common Expression Language (CEL).

Implementando um PEP

Um ponto de aplicação de políticas (PEP) é responsável por receber solicitações de autorização que são enviadas ao ponto de decisão de política (PDP) para avaliação. Um PEP pode estar em qualquer lugar em um aplicativo em que os dados e os recursos devem ser protegidos ou onde a lógica de autorização é aplicada. PEPs são relativamente simples em comparação com PDPs. Um PEP é responsável somente por solicitar e avaliar uma decisão de autorização e não exige nenhuma lógica de autorização. PEPs, ao contrário PDPs, não pode ser centralizado em um aplicativo SaaS. Isso ocorre porque a autorização e o controle de acesso precisam ser implementados em todo o aplicativo e em seus pontos de acesso. PEPs pode ser aplicado a microsserviços APIs, camadas de Backend for Frontend (BFF) ou a qualquer ponto do aplicativo em que o controle de acesso seja desejado ou necessário. A PEPs generalização em um aplicativo garante que a autorização seja verificada com frequência e de forma independente em vários pontos.

Para implementar um PEP, a primeira etapa é determinar onde a aplicação do controle de acesso deve ocorrer em um aplicativo. Considere esse princípio ao decidir onde PEPs deve ser integrado ao seu aplicativo:

Se um aplicativo expõe uma API, deve haver autorização e controle de acesso nessa API.

Isso ocorre porque, em uma arquitetura orientada a microserviços ou orientada a serviços, APIs servem como separadores entre diferentes funções do aplicativo. Faz sentido incluir o controle de acesso como pontos de verificação lógicos entre as funções do aplicativo. É altamente recomendável que você inclua PEPs como pré-requisito o acesso a cada API em um aplicativo SaaS. Também é possível integrar a autorização em outros pontos de um aplicativo. Em aplicações monolíticas, pode ser necessário PEPs integrá-las à lógica da própria aplicação. Não há um único local onde PEPs deva ser incluído, mas considere usar o princípio da API como ponto de partida.

Solicitando uma decisão de autorização

Um PEP deve solicitar uma decisão de autorização do PDP. A solicitação pode assumir várias formas. O método mais fácil e acessível para solicitar uma decisão de autorização é enviar uma solicitação ou consulta de autorização para uma RESTful API exposta pelo PDP (OPA ou Permissões Verificadas). Se você estiver usando Permissões verificadas, também poderá chamar o IsAuthorizedmétodo usando o AWS SDK para recuperar uma decisão de autorização. A única função de um PEP nesse padrão é encaminhar as informações de que a solicitação ou consulta de autorização precisa. Isso pode ser tão simples quanto encaminhar uma solicitação recebida por uma

API como entrada para o PDP. Existem outros métodos para criar PEPs. Por exemplo, você pode integrar um OPA PDP localmente com um aplicativo escrito na linguagem de programação Go como uma biblioteca em vez de usar uma API.

Avaliando uma decisão de autorização

PEPs precisam incluir lógica para avaliar os resultados de uma decisão de autorização. Quando PDPs são expostos como APIs, a decisão de autorização provavelmente está no formato JSON e é retornada por uma chamada de API. O PEP deve avaliar esse código JSON para determinar se a ação que está sendo tomada está autorizada. Por exemplo, se um PDP for projetado para fornecer uma decisão booleana de autorização de permissão ou negação, o PEP pode simplesmente verificar esse valor e retornar o código de status HTTP 200 para permitir e o código de status HTTP 403 para negar. Esse padrão de incorporação de um PEP como pré-requisito para acessar uma API é um padrão de fácil implementação e altamente eficaz para implementar o controle de acesso em um aplicativo SaaS. Em cenários mais complicados, o PEP pode ser responsável por avaliar o código JSON arbitrário retornado pelo PDP. O PEP deve ser escrito para incluir qualquer lógica necessária para interpretar a decisão de autorização que o PDP retorna. Como é provável que um PEP seja implementado em muitos lugares diferentes em seu aplicativo, recomendamos que você empacote seu código PEP como uma biblioteca ou artefato reutilizável na linguagem de programação de sua escolha. Dessa forma, seu PEP pode ser facilmente integrado em qualquer ponto do seu aplicativo com o mínimo de retrabalho.

Modelos de design para arquiteturas SaaS multilocatário

Há muitas maneiras de implementar o controle de acesso e a autorização da API. Este guia se concentra em três modelos de design que são eficazes para arquiteturas SaaS multilocatário. Esses projetos servem como uma referência de alto nível para a implementação de pontos de decisão política (PDPs) e pontos de aplicação de políticas (PEPs), para formar um modelo de autorização coeso e onipresente para aplicativos.

Modelos de design:

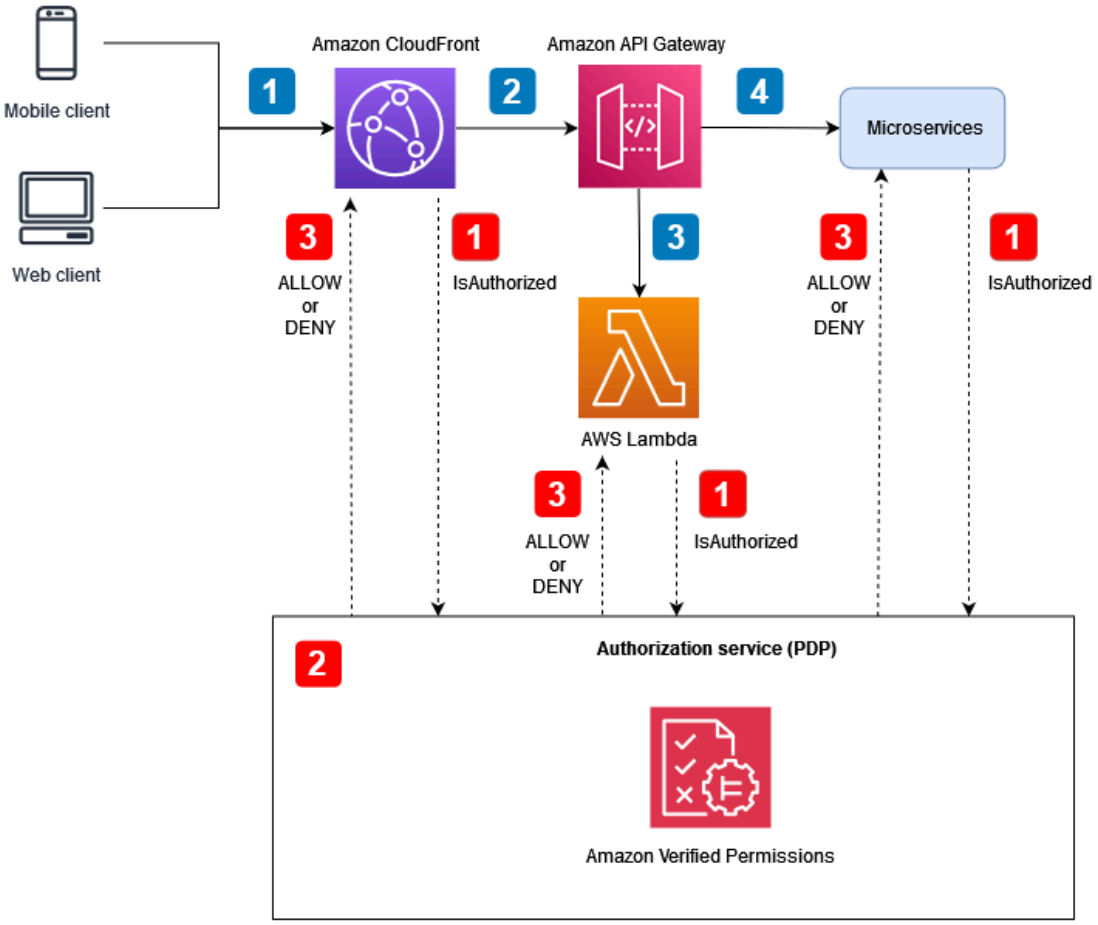
- Modelos de design para Amazon Verified Permissions
- Modelos de design para OPA

Modelos de design para Amazon Verified Permissions

Usando um PDP centralizado com um PEPs APIs

O ponto de decisão política (PDP) centralizado com pontos de aplicação de políticas (PEPs) no APIs modelo segue as melhores práticas do setor para criar um sistema eficaz e de fácil manutenção para controle e autorização de acesso à API. Essa abordagem apóia vários princípios fundamentais:

- A autorização e o controle de acesso à API são aplicados em vários pontos do aplicativo.
- A lógica de autorização é independente do aplicativo.
- As decisões de controle de acesso são centralizadas.



Fluxo do aplicativo (ilustrado com textos explicativos numerados em azul no diagrama):

1. Um usuário autenticado com um JSON Web Token (JWT) gera uma solicitação HTTP para a Amazon. CloudFront
2. CloudFront encaminha a solicitação para o Amazon API Gateway, que está configurado como CloudFront origem.
3. Um autorizador personalizado do API Gateway é chamado para verificar o JWT.
4. Os microsserviços respondem à solicitação.

Fluxo de autorização e controle de acesso à API (ilustrado com textos explicativos numerados em vermelho no diagrama):

1. O PEP chama o serviço de autorização e passa os dados da solicitação, incluindo qualquer JWTs um.

2. O serviço de autorização (PDP), nesse caso, Permissões verificadas, usa os dados da solicitação como entrada da consulta e os avalia com base nas políticas relevantes especificadas pela consulta.
3. A decisão de autorização é devolvida ao PEP e avaliada.

Esse modelo usa um PDP centralizado para tomar decisões de autorização. PEPs são implementados em diferentes pontos para fazer solicitações de autorização ao PDP. O diagrama a seguir mostra como você pode implementar esse modelo em um aplicativo SaaS multilocatário hipotético.

Nessa arquitetura, PEPs solicite decisões de autorização nos endpoints de serviço da Amazon CloudFront e do Amazon API Gateway e para cada microsserviço. A decisão de autorização é tomada pelo serviço de autorização Amazon Verified Permissions (o PDP). Como o Verified Permissions é um serviço totalmente gerenciado, você não precisa gerenciar a infraestrutura subjacente. Você pode interagir com as permissões verificadas usando uma RESTful API ou o AWS SDK.

Você também pode usar essa arquitetura com mecanismos de política personalizados. No entanto, todas as vantagens obtidas com as permissões verificadas devem ser substituídas pela lógica fornecida pelo mecanismo de políticas personalizadas.

Um PDP centralizado com PEPs on APIs fornece uma opção fácil de criar um sistema de autorização robusto para. APIs Isso simplifica o processo de autorização e também fornece uma easy-to-use interface repetível para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou outros componentes do aplicativo.

Usando o SDK do Cedar

O Amazon Verified Permissions usa a linguagem Cedar para gerenciar permissões refinadas em seus aplicativos personalizados. Com as permissões verificadas, você pode armazenar as políticas do Cedar em um local central, aproveitar a baixa latência com processamento em milissegundos e auditar permissões em diferentes aplicativos. Opcionalmente, você também pode integrar o SDK do Cedar diretamente ao seu aplicativo para fornecer decisões de autorização sem usar permissões verificadas. Essa opção requer desenvolvimento adicional de aplicativos personalizados para gerenciar e armazenar políticas para seu caso de uso. No entanto, pode ser uma alternativa viável, especialmente nos casos em que o acesso às Permissões Verificadas é intermitente ou não é possível devido à conectividade inconsistente com a Internet.

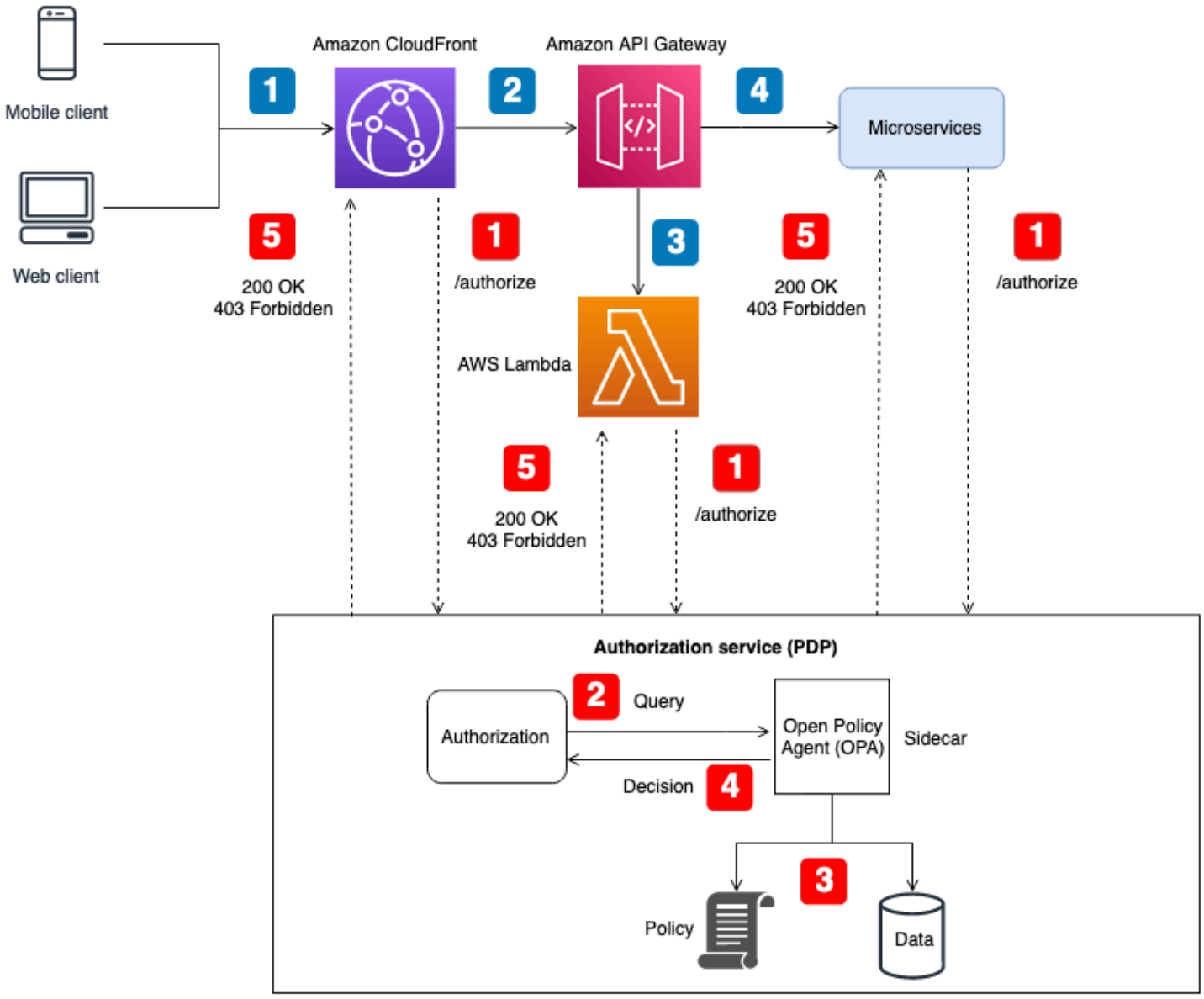
Modelos de design para OPA

Usando um PDP centralizado com um PEPs APIs

O ponto de decisão política (PDP) centralizado com pontos de aplicação de políticas (PEPs) no APIs modelo segue as melhores práticas do setor para criar um sistema eficaz e de fácil manutenção para controle e autorização de acesso à API. Essa abordagem apóia vários princípios fundamentais:

- A autorização e o controle de acesso à API são aplicados em vários pontos do aplicativo.
- A lógica de autorização é independente do aplicativo.
- As decisões de controle de acesso são centralizadas.

Esse modelo usa um PDP centralizado para tomar decisões de autorização. PEPs são implementados de todo APIs para fazer solicitações de autorização ao PDP. O diagrama a seguir mostra como você pode implementar esse modelo em um aplicativo SaaS multilocatário hipotético.



Fluxo do aplicativo (ilustrado com textos explicativos numerados em azul no diagrama):

1. Um usuário autenticado com um JWT gera uma solicitação HTTP para a Amazon. CloudFront
2. CloudFront encaminha a solicitação para o Amazon API Gateway, que está configurado como CloudFront origem.
3. Um autorizador personalizado do API Gateway é chamado para verificar o JWT.
4. Os microsserviços respondem à solicitação.

Fluxo de autorização e controle de acesso à API (ilustrado com textos explicativos numerados em vermelho no diagrama):

1. O PEP chama o serviço de autorização e passa os dados da solicitação, incluindo qualquer JWTs um.
2. O serviço de autorização (PDP) pega os dados da solicitação e consulta uma API REST do agente OPA, que está sendo executada como um sidecar. Os dados da solicitação servem como entrada para a consulta.
3. A OPA avalia a entrada com base nas políticas relevantes especificadas pela consulta. Os dados são importados para tomar uma decisão de autorização, se necessário.
4. A OPA devolve uma decisão ao serviço de autorização.
5. A decisão de autorização é devolvida ao PEP e avaliada.

Nessa arquitetura, PEPs solicite decisões de autorização nos endpoints de serviço da Amazon CloudFront e do Amazon API Gateway e para cada microsserviço. A decisão de autorização é tomada por um serviço de autorização (o PDP) com um sidecar da OPA. Você pode operar esse serviço de autorização como um contêiner ou como uma instância de servidor tradicional. O sidecar da OPA expõe sua RESTful API localmente para que a API seja acessível somente ao serviço de autorização. O serviço de autorização expõe uma API separada que está disponível para o. PEPs Fazer com que o serviço de autorização PEPs atue como intermediário entre uma OPA permite a inserção de qualquer lógica de transformação entre PEPs uma OPA que possa ser necessária, por exemplo, quando a solicitação de autorização de um PEP não está em conformidade com a entrada de consulta esperada pela OPA.

Você também pode usar essa arquitetura com mecanismos de política personalizados. No entanto, todas as vantagens obtidas com o OPA devem ser substituídas pela lógica fornecida pelo mecanismo de política personalizada.

Um PDP centralizado com PEPs on APIs fornece uma opção fácil de criar um sistema de autorização robusto para. APIs É simples de implementar e também fornece uma easy-to-use interface repetível para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou outros componentes do aplicativo. No entanto, essa abordagem pode criar muita latência em seu aplicativo, porque as decisões de autorização exigem a chamada de uma API separada. Se a latência da rede for um problema, você pode considerar uma PDP distribuída.

Usando um PDP distribuído com um PEPs APIs

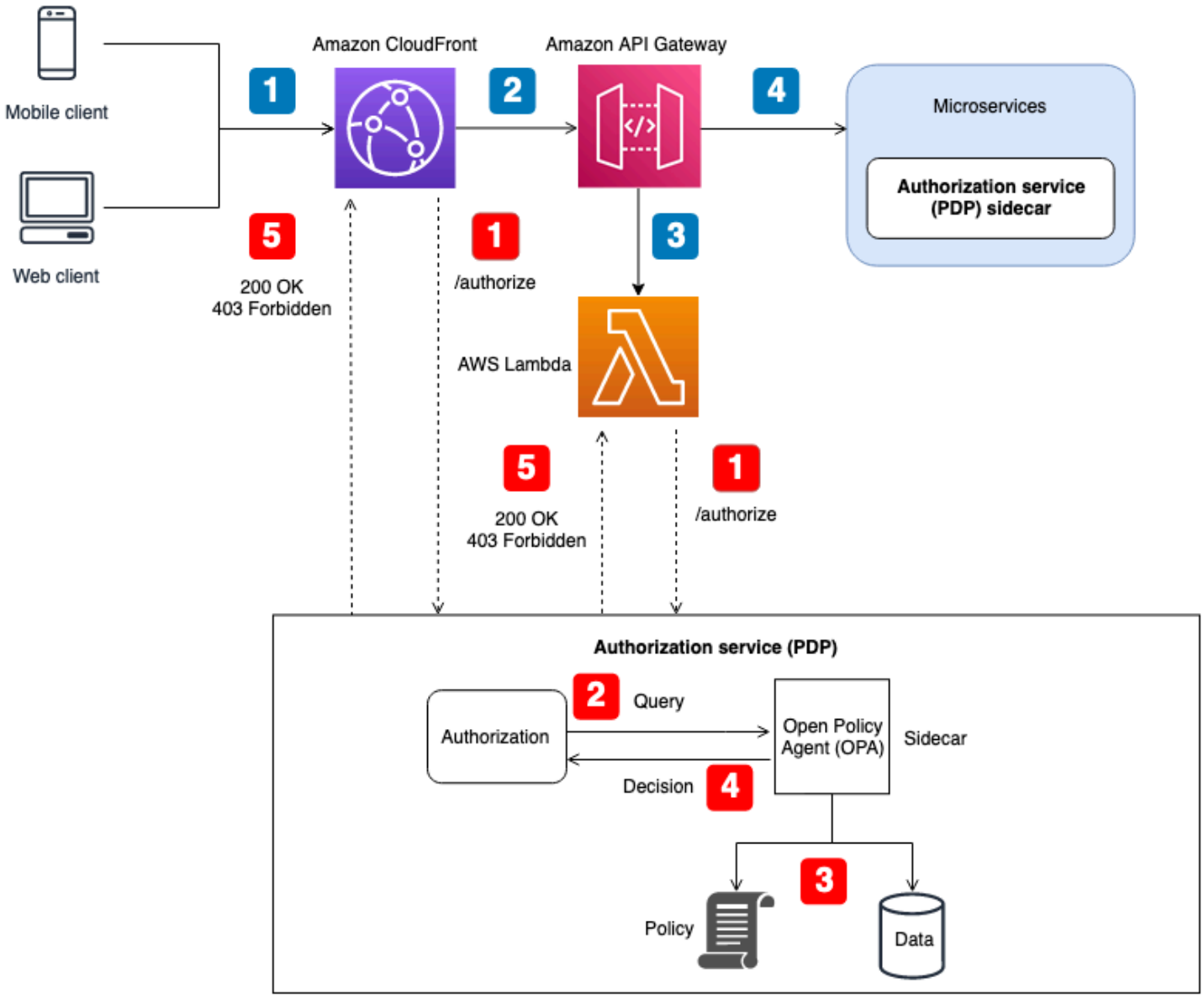
O ponto de decisão de política distribuído (PDP) com pontos de aplicação de políticas (PEPs) no APIs modelo segue as melhores práticas do setor para criar um sistema eficaz para controle e autorização de acesso à API. Assim como acontece com o PDP centralizado com PEPs um APIs modelo, essa abordagem suporta os seguintes princípios fundamentais:

- A autorização e o controle de acesso à API são aplicados em vários pontos do aplicativo.
- A lógica de autorização é independente do aplicativo.
- As decisões de controle de acesso são centralizadas.

Você pode se perguntar por que as decisões de controle de acesso são centralizadas quando o PDP é distribuído. Embora o PDP possa existir em vários locais em um aplicativo, ele deve usar a mesma lógica de autorização para tomar decisões de controle de acesso. Todos PDPs fornecem as mesmas decisões de controle de acesso com as mesmas entradas. PEPs são implementados de todo APIs para fazer solicitações de autorização ao PDP. A figura a seguir mostra como esse modelo distribuído pode ser implementado em um aplicativo SaaS multilocatário hipotético.

Nessa abordagem, os PDPs são implementados em vários locais no aplicativo. Para componentes de aplicativos que têm recursos computacionais integrados que podem executar OPA e oferecer suporte a um PDP, como um serviço em contêiner com um sidecar ou uma instância do Amazon Elastic Compute Cloud (Amazon EC2), as decisões de PDP podem ser integradas diretamente ao componente do aplicativo sem a necessidade de fazer uma chamada de API para um serviço de PDP centralizado. RESTful Isso tem a vantagem de reduzir a latência que você pode encontrar no modelo PDP centralizado, porque nem todo componente do aplicativo precisa fazer chamadas de API adicionais para obter decisões de autorização. No entanto, um PDP centralizado ainda é necessário nesse modelo para componentes de aplicativos que não têm recursos computacionais integrados que permitam a integração direta de um PDP, como o serviço Amazon ou Amazon CloudFront API Gateway.

O diagrama a seguir mostra como essa combinação de uma PDP centralizada e uma PDP distribuída pode ser implementada em um hipotético aplicativo SaaS multilocatário.



Fluxo do aplicativo (ilustrado com textos explicativos numerados em azul no diagrama):

1. Um usuário autenticado com um JWT gera uma solicitação HTTP para a Amazon. CloudFront
2. CloudFront encaminha a solicitação para o Amazon API Gateway, que está configurado como CloudFront origem.
3. Um autorizador personalizado do API Gateway é chamado para verificar o JWT.
4. Os microsserviços respondem à solicitação.

Fluxo de autorização e controle de acesso à API (ilustrado com textos explicativos numerados em vermelho no diagrama):

1. O PEP chama o serviço de autorização e passa os dados da solicitação, incluindo qualquer JWTs um.
2. O serviço de autorização (PDP) pega os dados da solicitação e consulta uma API REST do agente OPA, que está sendo executada como um sidecar. Os dados da solicitação servem como entrada para a consulta.
3. A OPA avalia a entrada com base nas políticas relevantes especificadas pela consulta. Os dados são importados para tomar uma decisão de autorização, se necessário.
4. A OPA devolve uma decisão ao serviço de autorização.
5. A decisão de autorização é devolvida ao PEP e avaliada.

Nessa arquitetura, os PEPs solicitam decisões de autorização nos endpoints de serviço do API Gateway CloudFront e de cada microsserviço. A decisão de autorização para microsserviços é feita por um serviço de autorização (o PDP) que opera como um sidecar com o componente do aplicativo. Esse modelo é possível para microsserviços (ou serviços) executados em contêineres ou instâncias do Amazon Elastic Compute Cloud (Amazon EC2). Decisões de autorização para serviços como o API Gateway ainda CloudFront exigiriam o contato com um serviço de autorização externo. Independentemente disso, o serviço de autorização expõe uma API que está disponível para o. PEPs Fazer com que o serviço de autorização PEPs atue como intermediário entre uma OPA permite a inserção de qualquer lógica de transformação entre PEPs uma OPA que possa ser necessária, por exemplo, quando a solicitação de autorização de um PEP não está em conformidade com a entrada de consulta esperada pela OPA.

Você também pode usar essa arquitetura com mecanismos de política personalizados. No entanto, todas as vantagens obtidas com o OPA devem ser substituídas pela lógica fornecida pelo mecanismo de política personalizada.

Um PDP distribuído com PEPs on APIs fornece uma opção para criar um sistema de autorização robusto para APIs. É simples de implementar e fornece uma easy-to-use interface repetível para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou outros componentes do aplicativo. Essa abordagem também tem a vantagem de reduzir a latência que você pode encontrar no modelo PDP centralizado.

Usando um PDP distribuído como biblioteca

Você também pode solicitar decisões de autorização de um PDP que é disponibilizado como uma biblioteca ou pacote para uso em um aplicativo. O OPA pode ser usado como uma biblioteca Go terceirizada. Para outras linguagens de programação, adotar esse modelo geralmente significa que você deve criar um mecanismo de política personalizado.

Considerações sobre o design multilocatário do Amazon Verified Permissions

Há várias opções de design a serem consideradas quando você implementa a autorização usando Amazon Verified Permissions em uma solução SaaS multilocatária. Antes de explorar essas opções, vamos esclarecer a diferença entre isolamento e autorização em um contexto SaaS multilocatário. O isolamento de um inquilino evita que os dados de entrada e saída sejam expostos ao inquilino errado. A autorização garante que o usuário tenha as permissões para acessar um inquilino.

Em Permissões verificadas, as políticas são armazenadas em um repositório de políticas. Conforme descrito na documentação de Permissões verificadas, você pode isolar as políticas dos inquilinos usando um repositório de políticas separado para cada inquilino ou permitir que os inquilinos compartilhem políticas usando um único repositório de políticas para todos os inquilinos. Esta seção discute as vantagens e desvantagens dessas duas estratégias de isolamento e descreve como elas podem ser implantadas usando um modelo de implantação em camadas. Para obter mais contexto, consulte a documentação de permissões verificadas.

Embora os critérios discutidos nesta seção se concentrem nas permissões verificadas, os conceitos gerais estão enraizados na mentalidade de isolamento e na orientação que ela fornece. Os aplicativos SaaS devem sempre considerar o isolamento do inquilino como parte de seu design, e esse princípio geral de isolamento se estende à inclusão de permissões verificadas em um aplicativo SaaS. Esta seção também faz referência aos principais modelos de isolamento de SaaS, como o modelo SaaS em silos e o modelo SaaS em pool. Para obter informações adicionais, consulte os principais conceitos de isolamento no AWS Well-Architected Framework, SaaS Lens.

As principais considerações ao projetar soluções SaaS multilocatário são o isolamento e a integração de inquilinos. O isolamento do inquilino afeta a segurança, a privacidade, a resiliência e o desempenho. A integração de inquilinos afeta seus processos operacionais no que se refere à sobrecarga operacional e à observabilidade. Organizações que passam por uma jornada de SaaS ou implementam soluções multilocatárias devem sempre priorizar como a locação será tratada pelo aplicativo SaaS. Embora uma solução SaaS possa se inclinar para um modelo de isolamento específico, a consistência não é necessariamente necessária em toda a solução SaaS. Por exemplo, o modelo de isolamento escolhido para os componentes de front-end do seu aplicativo pode não ser o mesmo que o modelo de isolamento escolhido para um microsserviço ou serviços de autorização.

Considerações de design:

Integração de inquilinos e registro de inquilinos de usuários

Os aplicativos SaaS observam o conceito de identidades SaaS e seguem a melhor prática geral de vincular uma identidade de usuário a uma identidade de inquilino. A vinculação envolve armazenar um identificador de inquilino como uma declaração ou atributo para o usuário no provedor de identidade. Isso transfere a responsabilidade de mapear as identidades dos inquilinos de cada aplicativo para o processo de registro do usuário. Cada usuário autenticado então tem a identidade correta do locatário como parte do JSON Web Token (JWT).

Da mesma forma, a seleção do repositório de políticas correto para uma solicitação de autorização não deve ser determinada pela lógica do aplicativo. Para determinar qual repositório de políticas uma solicitação de autorização específica deve usar, mantenha um mapeamento dos usuários para os repositórios de políticas ou dos inquilinos para os repositórios de políticas. Esses mapeamentos geralmente são mantidos em um armazenamento de dados, como o Amazon DynamoDB ou o Amazon Relational Database Service (Amazon RDS) ao qual seu aplicativo faz referência. Você também pode fornecer ou complementar esses mapeamentos por dados em um provedor de identidade (IdP). O relacionamento entre inquilinos, usuários e repositórios de políticas geralmente é fornecido a um usuário por meio de um JWT que contém todos os relacionamentos necessários para uma solicitação de autorização.

Este exemplo mostra como o JWT pode aparecer para o usuário`Alice`, que pertence ao locatário `TenantA` e usa o repositório de políticas com o ID do repositório de políticas `ps-43214321` para autorização.

```
{
   "sub":"1234567890",
   "name":"Alice",
   "tenant":"TenantA",
   "policyStoreId":"ps-43214321"
}
```

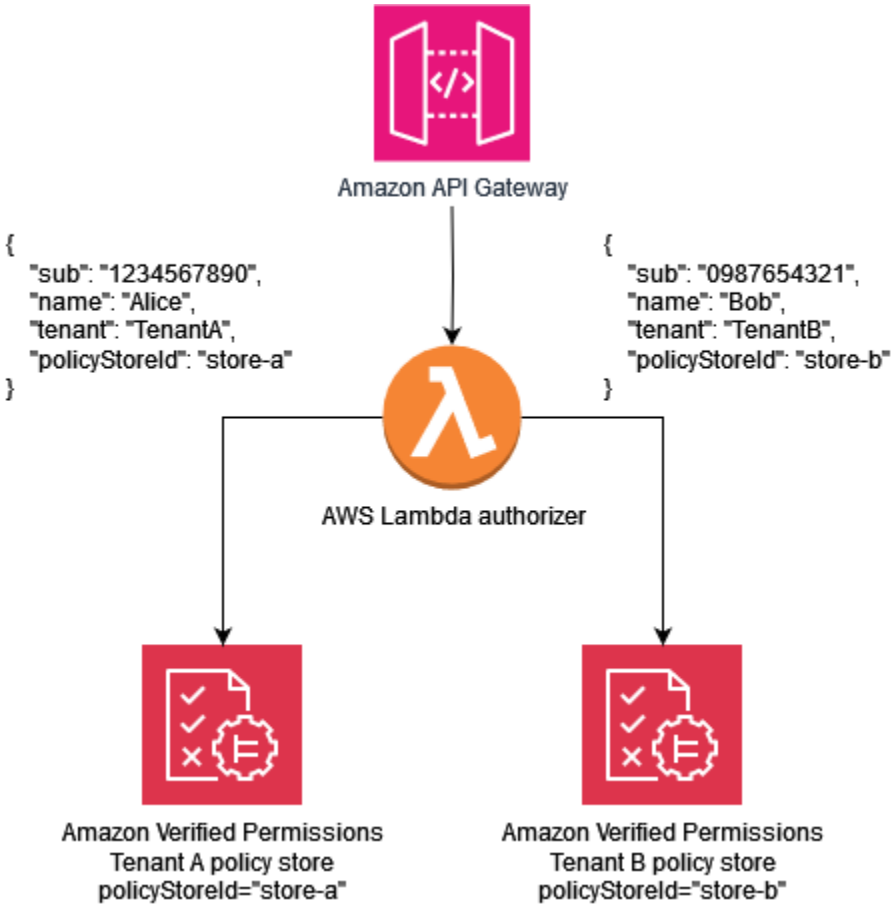
Armazenamento de políticas por inquilino

O modelo de design do repositório de políticas por inquilino no Amazon Verified Permissions associa cada inquilino em um aplicativo SaaS ao seu próprio armazenamento de políticas. Esse modelo é semelhante ao modelo de isolamento de silos SaaS. Ambos os modelos exigem a criação de infraestrutura específica para inquilinos e têm vantagens e desvantagens semelhantes. Os principais benefícios dessa abordagem são o isolamento de inquilinos imposto pela infraestrutura, o suporte a modelos de autorização exclusivos por locatário, a eliminação de preocupações ruidosas com vizinhos e um escopo reduzido de impacto de falhas nas atualizações ou implantações de políticas. As desvantagens dessa abordagem incluem processos, implantações e operações mais complexos de integração de inquilinos. O armazenamento de políticas por inquilino é a abordagem recomendada se a solução tiver políticas exclusivas por inquilino.

O modelo de armazenamento de políticas por inquilino pode fornecer uma abordagem altamente isolada para o isolamento de inquilinos, se seu aplicativo SaaS exigir. Você também pode usar esse modelo com isolamento de pool, mas sua implementação de Permissões Verificadas não compartilhará os benefícios padrão do modelo mais amplo de isolamento de pool, como gerenciamento e operações simplificados.

Em um repositório de políticas por inquilino, o isolamento do inquilino é obtido mapeando o identificador do repositório de políticas do inquilino para a identidade SaaS do usuário durante o processo de registro do usuário, conforme discutido anteriormente. Essa abordagem vincula fortemente o armazenamento de políticas do locatário ao usuário principal e fornece uma maneira consistente de compartilhar o mapeamento em toda a solução SaaS. Você pode fornecer o mapeamento para um aplicativo SaaS mantendo-o como parte de um IdP ou em uma fonte de dados externa, como o DynamoDB. Isso também garante que o diretor faça parte do inquilino e que o repositório de políticas do inquilino seja usado.

Este exemplo mostra como o JWT que contém a `policyStoreId` e `tenant` de um usuário é passado do endpoint da API para o ponto de avaliação da política em um AWS Lambda autorizador, que encaminha a solicitação para o repositório de políticas correto.



O exemplo de política a seguir ilustra o paradigma de design do repositório de políticas por inquilino. O usuário `Alice` pertence ao `TenantA.` The também policyStoreId `store-a` é mapeado para a identidade do inquilino `Alice,` e impõe o uso do repositório de políticas correto. Isso garante que as políticas de `TenantA` sejam usadas.

> **Note**
>
> O modelo de armazenamento de políticas por inquilino isola as políticas dos inquilinos. A autorização impõe as ações que os usuários podem realizar em seus dados. Os recursos envolvidos em qualquer aplicativo hipotético que usa esse modelo devem ser isolados usando outros mecanismos de isolamento, conforme definido na documentação do AWS Well-Architected Framework, SaaS Lens.

Nesta política, `Alice` tem permissões para visualizar os dados de todos os recursos.

```
permit (
```

```
    principal == MultiTenantApp::User::"Alice",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

Para fazer uma solicitação de autorização e iniciar uma avaliação com uma política de permissões verificadas, você precisa fornecer o ID do repositório de políticas que corresponde ao ID exclusivo mapeado para o inquilino,. `store-a`

```
{
   "policyStoreId":"store-a",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Alice"
               },
               "attributes":{},
               "parents":[]
            },
            {
               "identifier":{
                  "entityType":"MultiTenantApp::Data",
                  "entityId":"my_example_data"
               },
               "attributes":{},
               "parents":[]
            }
```

```
         ]
      ]
   }
}
```

O usuário Bob pertence ao Locatário B e também policyStoreId `store-b` é mapeado para a identidade do locatárioBob, o que impõe o uso do repositório de políticas correto. Isso garante que as políticas do Locatário B sejam usadas.

Nesta política, Bob tem permissões para personalizar os dados de todos os recursos. Neste exemplo, `customizeData` pode ser uma ação específica somente para o Locatário B, portanto, a política seria exclusiva para o Locatário B. O modelo de armazenamento de políticas por inquilino suporta inerentemente políticas personalizadas por inquilino.

```
permit (
    principal == MultiTenantApp::User::"Bob",
    action == MultiTenantApp::Action::"customizeData",
    resource
);
```

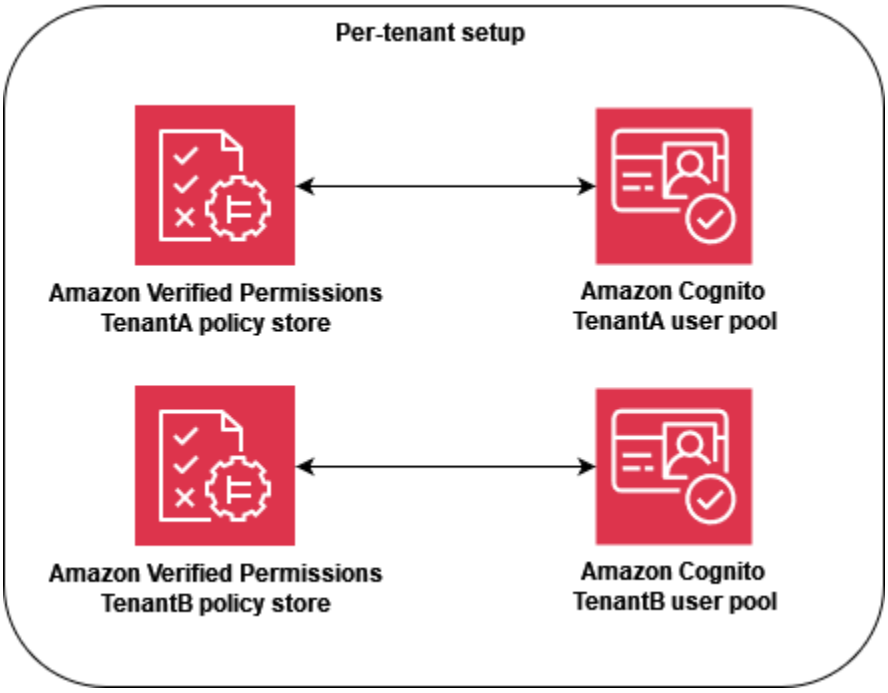
Para fazer uma solicitação de autorização e iniciar uma avaliação com uma política de permissões verificadas, você precisa fornecer o ID do repositório de políticas que corresponde ao ID exclusivo mapeado para o inquilino,. `store-b`

```
{
   "policyStoreId":"store-b",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Bob"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"customizeData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         [
```

```
            {
               "identifier":{
                  "entityType":"MultiTenantApp::User",
                  "entityId":"Bob"
               },
               "attributes":{},
               "parents":[]
            },
            {
               "identifier":{
                  "entityType":"MultiTenantApp::Data",
                  "entityId":"my_example_data"
               },
               "attributes":{},
               "parents":[]
            }
         ]
      ]
   }
}
```

Com as Permissões Verificadas, é possível, mas não obrigatório, integrar um IdP a um repositório de políticas. Essa integração permite que as políticas referenciem explicitamente o principal no repositório de identidades como o principal das políticas. Para obter mais informações sobre como se integrar ao Amazon Cognito como um IdP para permissões verificadas, consulte a documentação de permissões verificadas e a documentação do Amazon Cognito.

Ao integrar um repositório de políticas a um IdP, você pode usar somente uma fonte de identidade por repositório de políticas. Por exemplo, se você optar por integrar as Permissões Verificadas com o Amazon Cognito, precisará espelhar a estratégia usada para o isolamento de locatários dos repositórios de políticas de Permissões Verificadas e dos grupos de usuários do Amazon Cognito. Os repositórios de políticas e os grupos de usuários também precisam estar nos mesmos Conta da AWS.



Em um nível operacional, o armazenamento de políticas por inquilino tem uma vantagem de auditoria, porque você pode consultar facilmente a atividade registrada de formaAWS CloudTrail independente para cada inquilino. No entanto, ainda recomendamos que você registre métricas personalizadas adicionais em uma dimensão por inquilino na Amazon. CloudWatch

A abordagem de armazenamento de políticas por inquilino também exige muita atenção a duas cotas de permissões verificadas para garantir que elas não interfiram nas operações de sua solução SaaS. Essas cotas são repositórios de políticas por região por conta e IsAuthorized solicitações por segundo por região por conta. Você pode solicitar aumentos para ambas as cotas.

Para um exemplo mais detalhado de como implementar o modelo de armazenamento de políticas por inquilino, consulte a AWS postagem no blog Controle de acesso SaaS usando Amazon Verified Permissions com um armazenamento de políticas por inquilino.

Um repositório de políticas compartilhado para vários locatários

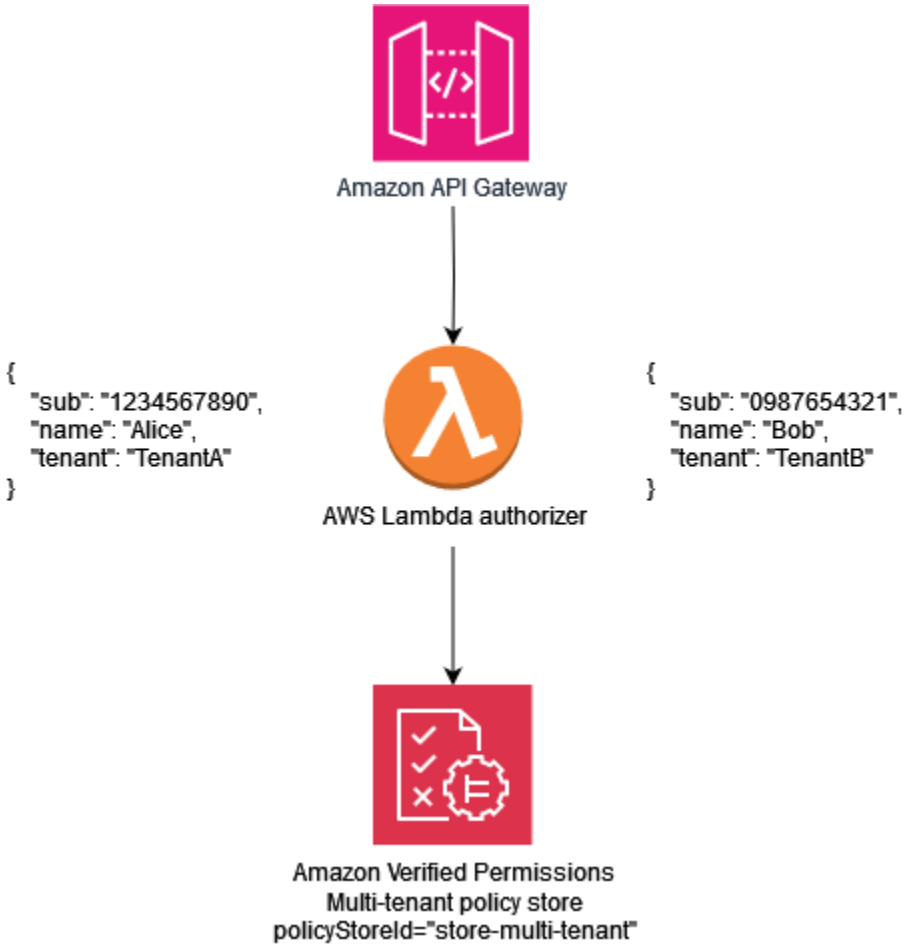
O modelo de design de um repositório de políticas multilocatário compartilhado usa um único repositório de políticas multilocatário no Amazon Verified Permissions para todos os inquilinos na solução SaaS. O principal benefício dessa abordagem é o gerenciamento e as operações simplificados, principalmente porque você não precisa criar repositórios de políticas adicionais durante a integração do inquilino. As desvantagens dessa abordagem incluem um maior escopo

de impacto de qualquer falha ou erro nas atualizações ou implantações de políticas e uma maior exposição aos efeitos ruidosos da vizinhança. Além disso, não recomendamos essa abordagem se sua solução exigir políticas exclusivas para cada inquilino. Nesse caso, use o modelo de armazenamento de políticas por inquilino para garantir que as políticas do inquilino correto sejam usadas.

A única abordagem compartilhada de armazenamento de políticas multilocatário é semelhante ao modelo de isolamento agrupado de SaaS. Ele pode fornecer uma abordagem agrupada para o isolamento de inquilinos, se seu aplicativo SaaS exigir. Você também pode usar esse modelo se sua solução SaaS aplicar isolamento em silos aos microsserviços. Ao escolher um modelo, você deve avaliar os requisitos de isolamento de dados do inquilino e a estrutura das políticas de Permissões Verificadas que são necessárias para um aplicativo SaaS de forma independente.

Para impor uma forma consistente de compartilhar o identificador do locatário em toda a sua solução SaaS, é uma boa prática mapear o identificador para a identidade SaaS do usuário durante o registro do usuário, conforme discutido anteriormente. Você pode fornecer esse mapeamento para um aplicativo SaaS mantendo-o como parte de um IdP ou em uma fonte de dados externa, como o DynamoDB. Também recomendamos que você mapeie a ID do repositório de políticas compartilhadas para os usuários. Embora o ID não seja usado como parte do isolamento do inquilino, essa é uma boa prática porque facilita futuras mudanças.

O exemplo a seguir mostra como o endpoint da API envia um JWT para os usuários `Alice` eBob, que pertencem a diferentes locatários, mas compartilham o repositório de políticas com o ID `store-multi-tenant` do repositório de políticas para autorização. Como todos os locatários compartilham um único repositório de políticas, você não precisa manter o ID do repositório de políticas em um token ou banco de dados. Como todos os locatários compartilham uma única ID do repositório de políticas, você pode fornecer o ID como uma variável de ambiente que seu aplicativo pode usar para fazer chamadas para o repositório de políticas.



O exemplo de política a seguir ilustra o paradigma de design de uma política compartilhada para vários locatários. Nessa política, o diretor `MultiTenantApp::User` que tem o pai `MultiTenantApp::Role Admin` tem permissões para visualizar os dados de todos os recursos.

```
permit (
    principal in MultiTenantApp::Role::"Admin",
    action == MultiTenantApp::Action::"viewData",
    resource
);
```

Como um único repositório de políticas está em uso, o repositório de políticas de Permissões Verificadas deve garantir que um atributo de locação associado ao principal corresponda ao atributo de locação associado ao recurso. Isso pode ser feito incluindo a política a seguir no repositório de políticas, para garantir que todas as solicitações de autorização que não tenham atributos de locação correspondentes no recurso e no principal sejam rejeitadas.

```
forbid(
```

```
    principal,
    action,
    resource
)
unless {
    resource.Tenant == principal.Tenant
};
```

Para uma solicitação de autorização que usa um modelo de repositório de políticas multilocatário compartilhado, o ID do repositório de políticas é o identificador do repositório de políticas compartilhado. Na solicitação a seguir, o acesso `User Alice` é permitido porque ela tem um `Role` de`Admin`, e os `Tenant` atributos associados ao recurso e ao principal são ambos`TenantA`.

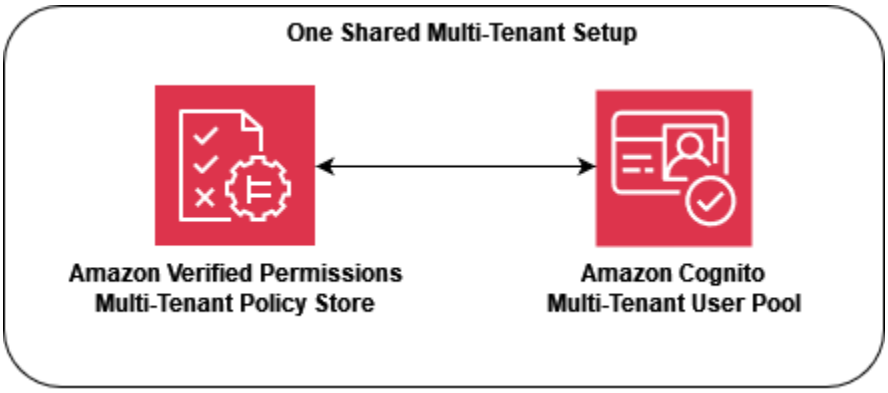
```
{
   "policyStoreId":"store-multi-tenant",
   "principal":{
      "entityType":"MultiTenantApp::User",
      "entityId":"Alice"
   },
   "action":{
      "actionType":"MultiTenantApp::Action",
      "actionId":"viewData"
   },
   "resource":{
      "entityType":"MultiTenantApp::Data",
      "entityId":"my_example_data"
   },
   "entities":{
      "entityList":[
         {
            "identifier":{
               "entityType":"MultiTenantApp::User",
               "entityId":"Alice"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
```

```
                }
            },
            "parents":[
               {
                  "entityType":"MultiTenantApp::Role",
                  "entityId":"Admin"
               }
            ]
         },
         {
            "identifier":{
               "entityType":"MultiTenantApp::Data",
               "entityId":"my_example_data"
            },
            "attributes": {
                {
                    "Tenant": {
                        "entityIdentifier": {
                            "entityType":"MultitenantApp::Tenant",
                            "entityId":"TenantA"
                        }
                    }
                }
            },
            "parents":[]
         }
      ]
   }
}
```

Com as Permissões Verificadas, é possível, mas não obrigatório, integrar um IdP a um repositório de políticas. Essa integração permite que as políticas referenciem explicitamente o principal no repositório de identidades como o principal das políticas. Para obter mais informações sobre como se integrar ao Amazon Cognito como um IdP para permissões verificadas, consulte a documentação de permissões verificadas e a documentação do Amazon Cognito.

Ao integrar um repositório de políticas a um IdP, você pode usar somente uma fonte de identidade por repositório de políticas. Por exemplo, se você optar por integrar as Permissões Verificadas com o Amazon Cognito, precisará espelhar a estratégia usada para o isolamento de locatários dos repositórios de políticas de Permissões Verificadas e dos grupos de usuários do Amazon Cognito.

Os repositórios de políticas e os grupos de usuários também precisam estar nos mesmos Conta da AWS.



Do ponto de vista operacional e de auditoria, o único modelo de armazenamento de políticas multilocatário compartilhado tem a desvantagem de que a atividade registrada AWS CloudTrail exige consultas mais complexas para filtrar a atividade individual do inquilino, porque cada chamada registrada CloudTrail usa o mesmo repositório de políticas. Nesse cenário, é útil registrar métricas personalizadas adicionais em uma dimensão por inquilino na Amazon CloudWatch para garantir um nível adequado de capacidade de observação e auditoria.

A abordagem compartilhada de armazenamento de políticas multilocatário também exige muita atenção às cotas de permissões verificadas para garantir que elas não interfiram nas operações de sua solução SaaS. Em particular, recomendamos que você monitore as IsAuthorized solicitações por segundo por região por cota de conta para garantir que suas limitações não sejam excedidas. Você pode solicitar um aumento dessa cota.

Modelo de implantação hierárquica

Ao criar um modelo de implantação em camadas, você pode isolar locatários de “nível corporativo” de alta prioridade do volume potencialmente maior de clientes de “nível padrão”. Nesse modelo, você pode implementar todas as alterações implantadas nas políticas nos repositórios de políticas separadamente para cada nível, o que isola cada nível de clientes das alterações feitas fora do nível. No modelo de implantação em camadas, os repositórios de políticas geralmente são criados como parte do provisionamento inicial da infraestrutura para cada nível, em vez de serem implantados quando um inquilino é integrado.

Se sua solução usa principalmente um modelo de isolamento em pool, você pode precisar de isolamento ou personalização adicionais. Por exemplo, você pode criar um “nível Premium” em

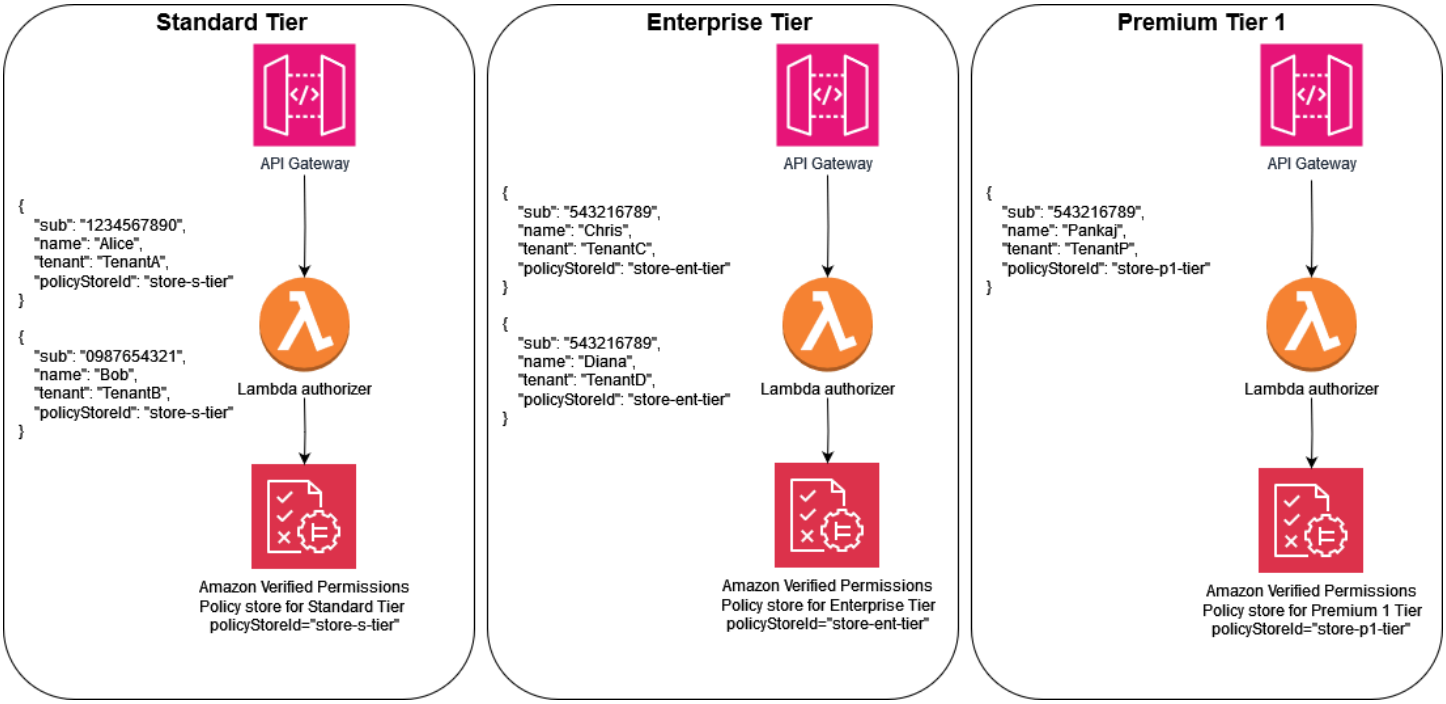
que cada locatário teria sua própria infraestrutura de nível de inquilino, o que cria um modelo em silos ao implantar uma instância em pool com apenas um inquilino. Isso pode assumir a forma de infraestruturas de “Locatário A de Nível Premium” e “Locatário de Nível Premium B” que são completamente separadas, incluindo repositórios de políticas. Essa abordagem resulta em um modelo de isolamento em silos para o mais alto nível de clientes.

No modelo de implantação em camadas, cada repositório de políticas deve seguir o mesmo modelo de isolamento, embora seja implantado separadamente. Como há vários repositórios de políticas sendo usados, você precisa aplicar uma maneira consistente de compartilhar o identificador do repositório de políticas associado ao locatário em toda a solução SaaS. Assim como no modelo de armazenamento de políticas por locatário, é uma boa prática mapear o identificador do inquilino para a identidade SaaS do usuário durante o registro do usuário.

O diagrama a seguir mostra três camadas: `Standard Tier`Enterprise Tier, e. `Premium Tier` 1 Cada camada é implantada separadamente em sua própria infraestrutura e usa um repositório de políticas compartilhado dentro da camada. Os níveis Standard e Enterprise contêm vários inquilinos. `TenantA`e `TenantB` estão no`Standard Tier`, e `TenantC` e `TenantD` estão no nível Enterprise.

`Premium Tier 1`contém somente`TenantP`, para que você possa atender ao inquilino premium como se a solução tivesse um modelo de isolamento totalmente isolado e fornecer recursos como políticas personalizadas. A integração de um novo cliente de nível premium resultaria na criação de uma `Premium Tier 2` infraestrutura.

> **Note**
>
> O aplicativo, a implantação e a integração de locatários no nível premium são idênticos aos níveis padrão e corporativo. A única diferença é que o fluxo de trabalho de integração de nível premium começa com o provisionamento de uma nova infraestrutura de nível.


Standard Tier
API Gateway
{
"sub": "1234567890",
"name": "Alice",
"tenant": "TenantA",
"policyStoreId": "store-s-tier"
}
{
"sub": "0987654321",
"name": "Bob",
"tenant": "TenantB",
"policyStoreId": "store-s-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Standard Tier
policyStoreId="store-s-tier"
Enterprise Tier
API Gateway
{
"sub": "543216789",
"name": "Chris",
"tenant": "TenantC",
"policyStoreId": "store-ent-tier"
}
{
"sub": "543216789",
"name": "Diana",
"tenant": "TenantD",
"policyStoreId": "store-ent-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Enterprise Tier
policyStoreId="store-ent-tier"
Premium Tier 1
API Gateway
{
"sub": "543216789",
"name": "Pankaj",
"tenant": "TenantP",
"policyStoreId": "store-p1-tier"
}
Lambda authorizer
Amazon Verified Permissions
Policy store for Premium 1 Tier
policyStoreId="store-p1-tier"

Considerações sobre o design multilocatário do OPA

O Open Policy Agent (OPA) é um serviço flexível que pode ser aplicado a vários casos de uso em que os aplicativos são necessários para tomar decisões políticas e de autorização. Usar o OPA com aplicativos SaaS multilocatários exige a consideração de critérios exclusivos para garantir que as principais práticas recomendadas de SaaS, como isolamento de inquilinos, continuem fazendo parte da implementação do OPA. Esses critérios incluem padrões de implantação do OPA, isolamento do inquilino e o modelo de documento do OPA e integração do inquilino. Cada um deles afeta o design ideal do OPA no que diz respeito a aplicativos multilocatários.

Embora a discussão nesta seção se concentre na OPA, os conceitos gerais estão enraizados na mentalidade de isolamento e na orientação que ela fornece. Os aplicativos SaaS devem sempre considerar o isolamento do inquilino como parte de seu design, e esse princípio geral de isolamento se estende à inclusão do OPA em um aplicativo SaaS. O OPA, se usado adequadamente, pode ser uma parte fundamental de como o isolamento é imposto em aplicativos SaaS. Esta seção também faz referência aos principais modelos de isolamento de SaaS, como o modelo SaaS em silos e o modelo SaaS em pool. Para obter informações adicionais, consulte os principais conceitos de isolamento no AWS Well-Architected Framework, SaaS Lens.

Considerações de design:

- Comparando padrões de implantação centralizada e distribuída
- Isolamento de inquilinos com o modelo de documento OPA
- Integração de inquilinos

Comparando padrões de implantação centralizada e distribuída

Você pode implantar o OPA em um padrão de implantação centralizado ou distribuído, e o método ideal para um aplicativo multilocatário depende do caso de uso. Para obter exemplos desses padrões, consulte as APIs seções Usando uma PDP centralizada com uma PDP PEPs ativada APIs e Usando uma PDP distribuída e PEPs ativada anteriormente neste guia. Como o OPA pode ser implantado como um daemon em um sistema operacional ou contêiner, ele pode ser implementado de várias maneiras para oferecer suporte a um aplicativo multilocatário.

Em um padrão de implantação centralizado, o OPA é implantado como um contêiner ou daemon com sua RESTful API disponível para outros serviços no aplicativo. Quando um serviço exige uma

decisão da OPA, a RESTful API central da OPA é chamada para produzir essa decisão. Essa abordagem é simples de implantar e manter, porque há apenas uma única implantação do OPA. A desvantagem dessa abordagem é que ela não fornece nenhum mecanismo para manter a separação dos dados do inquilino. Como há apenas uma única implantação da OPA, todos os dados do inquilino usados em uma decisão de OPA, incluindo quaisquer dados externos referenciados pela OPA, devem estar disponíveis para a OPA. Você pode manter o isolamento dos dados do inquilino com essa abordagem, mas ela deve ser aplicada pela política e pela estrutura de documentos da OPA ou pelo acesso a dados externos. Um padrão de implantação centralizado também exige uma latência maior, porque cada decisão de autorização deve fazer uma chamada de RESTful API para outro serviço.

Em um padrão de implantação distribuída, o OPA é implantado como um contêiner ou daemon junto com os serviços do aplicativo multilocatário. Ele pode ser implantado como um contêiner auxiliar ou como um daemon executado localmente no sistema operacional. Para recuperar uma decisão da OPA, o serviço faz uma chamada de RESTful API para a implantação local da OPA. (Como o OPA pode ser implantado como um pacote Go, você pode usar o Go nativamente para recuperar uma decisão em vez de usar uma RESTful chamada de API.) Diferentemente do padrão de implantação centralizada, o padrão distribuído exige um esforço muito mais robusto de implantação, manutenção e atualização, pois está presente em várias áreas do aplicativo. Um benefício do padrão de implantação distribuída é a capacidade de manter o isolamento dos dados do inquilino, especialmente para aplicativos que usam um modelo SaaS em silos. Os dados específicos do locatário podem ser isolados em implantações de OPA específicas desse locatário, porque a OPA em um modelo distribuído é implantada junto com o locatário. Além disso, um padrão de implantação distribuída tem uma latência muito menor do que um padrão de implantação centralizada, pois cada decisão de autorização pode ser tomada localmente.

Ao escolher um padrão de implantação de OPA em seu aplicativo multilocatário, certifique-se de avaliar os critérios mais críticos para seu aplicativo. Se seu aplicativo multilocatário for sensível à latência, um padrão de implantação distribuído oferece melhor desempenho às custas de implantação e manutenção mais complexas. Embora você possa gerenciar parte dessa complexidade por meio DevOps da automação, ela ainda exige um esforço adicional quando comparado a um padrão de implantação centralizado.

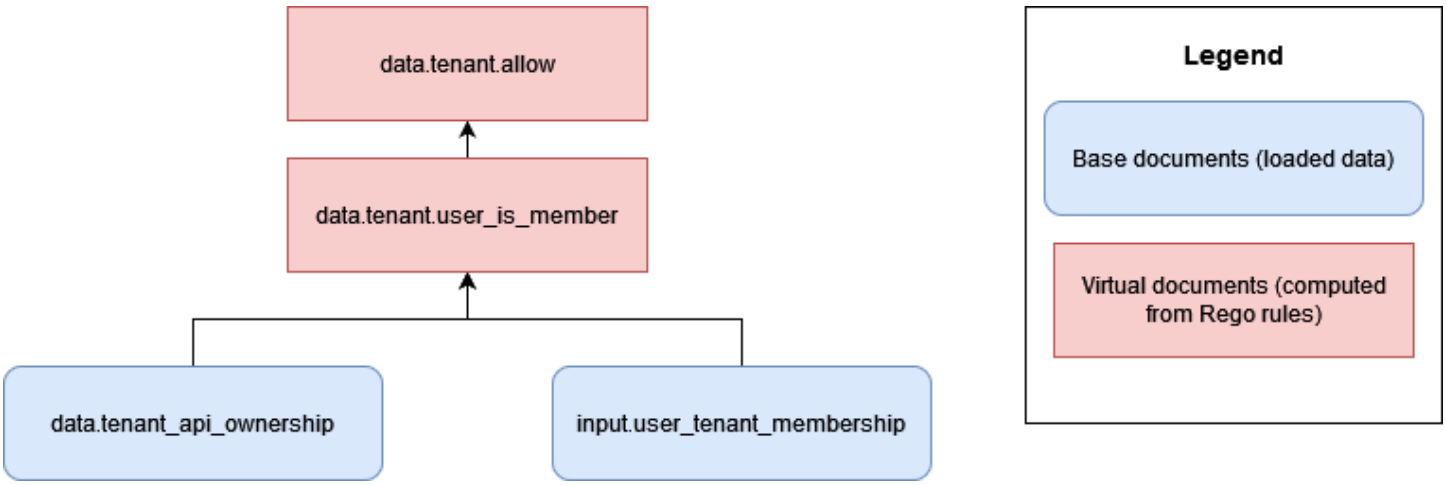
Se seu aplicativo multilocatário usa um modelo SaaS em silos, você pode usar um padrão de implantação de OPA distribuído para imitar a abordagem em silos do isolamento de dados do inquilino. Isso ocorre porque, quando o OPA é executado junto com cada serviço de aplicativo específico do inquilino, você pode personalizar cada implantação do OPA para conter apenas dados

associados a esse inquilino. Não é possível armazenar dados de OPA em um padrão centralizado de implantação de OPA. Se você usar um padrão de implantação centralizado ou um padrão distribuído em conjunto com um modelo SaaS agrupado, o isolamento dos dados do inquilino deverá ser mantido no modelo de documento OPA.

Isolamento de inquilinos com o modelo de documento OPA

A OPA usa documentos para tomar decisões. Esses documentos podem conter dados específicos do inquilino, portanto, você deve considerar como manter o isolamento dos dados do inquilino. Os documentos OPA consistem em documentos básicos e documentos virtuais. Os documentos básicos contêm dados do mundo exterior. Isso inclui dados fornecidos diretamente à OPA, dados sobre a solicitação de OPA e dados que podem ser passados para a OPA como entrada. Os documentos virtuais são computados por política e incluem políticas e regras de OPA. Para obter mais informações, consulte a documentação do OPA.

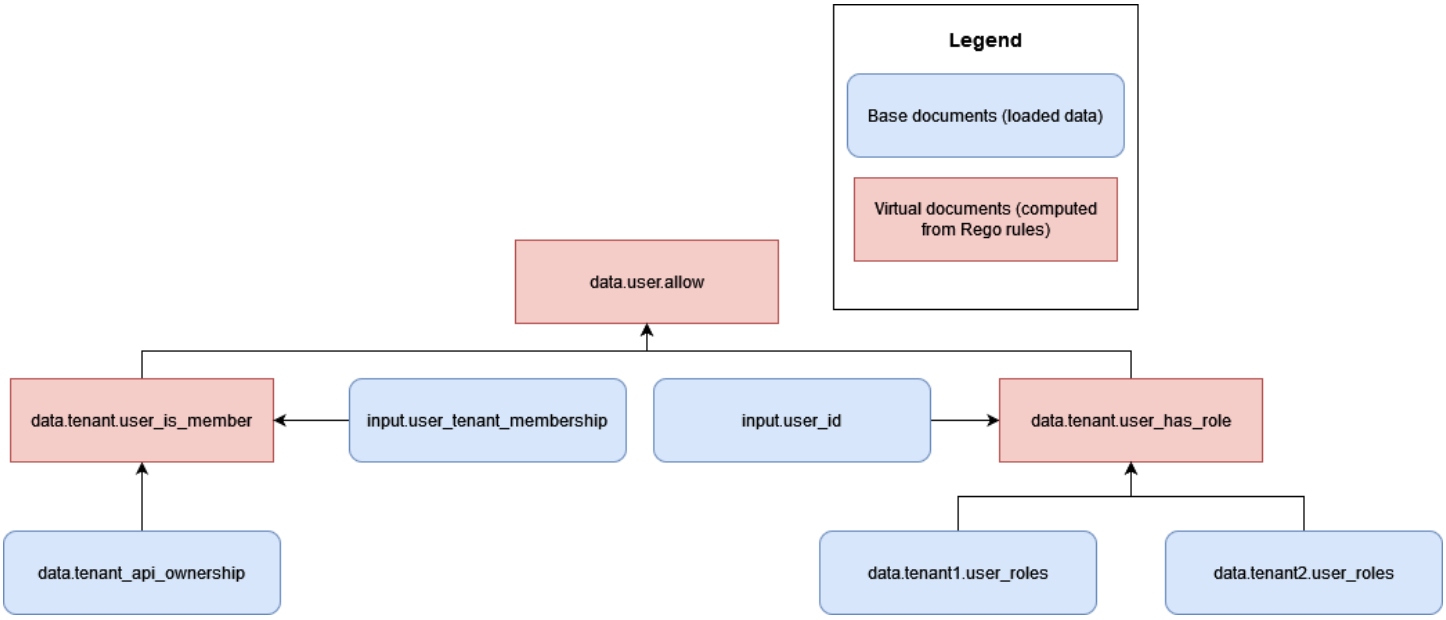
Para criar um modelo de documento no OPA para um aplicativo multilocatário, você deve primeiro considerar o tipo de documentos básicos necessários para tomar uma decisão no OPA. Se esses documentos básicos contiverem dados específicos do inquilino, você deverá tomar medidas para garantir que esses dados não sejam expostos acidentalmente ao acesso entre inquilinos. Felizmente, em muitos casos, dados específicos do inquilino não são necessários para tomar uma decisão na OPA. O exemplo a seguir mostra um modelo hipotético de documento OPA que permite acesso a uma API com base em qual inquilino é proprietário da API e se o usuário é membro do locatário, conforme indicado no documento de entrada.



Nessa abordagem, a OPA não tem acesso a nenhum dado específico do inquilino, exceto às informações sobre quais inquilinos possuem uma API. Nesse caso, não há preocupação com o fato de a OPA facilitar o acesso entre inquilinos, porque as únicas informações que a OPA usa para

tomar uma decisão de acesso são a associação do usuário com um inquilino e a associação do inquilino com. APIs Você poderia aplicar esse tipo de modelo de documento OPA a um modelo SaaS em silos, porque cada inquilino teria propriedade de recursos independentes.

No entanto, em muitas abordagens de autorização do RBAC, existe o potencial de exposição de informações entre inquilinos. No exemplo a seguir, um modelo hipotético de documento OPA permite o acesso a uma API com base no fato de o usuário ser membro de um locatário e se o usuário tem a função correta para acessar a API.



Esse modelo introduz o risco de acesso entre inquilinos, porque as funções e permissões de vários inquilinos estão e agora `data.tenant2.user_roles` devem ser disponibilizadas à OPA para tomar decisões de autorização. `data.tenant1.user_roles` Para manter o isolamento do inquilino e a privacidade do mapeamento de funções, esses dados não devem residir na OPA. Os dados do RBAC devem residir em uma fonte de dados externa, como um banco de dados. Além disso, o OPA não deve ser usado para mapear funções predefinidas para permissões específicas, pois isso dificulta que os inquilinos definam suas próprias funções e permissões. Isso também torna sua lógica de autorização rígida e precisa de atualização constante. Para obter orientação sobre como incorporar com segurança os dados do RBAC no processo de tomada de decisão da OPA, consulte a seção Recomendações para isolamento de inquilinos e privacidade de dados, mais adiante neste guia.

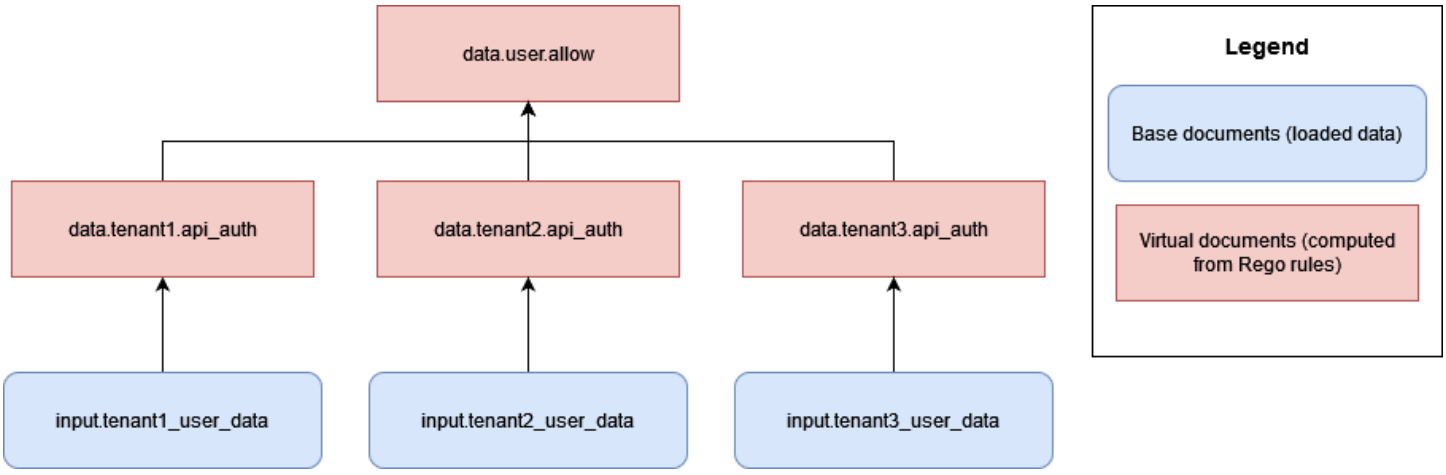
Você pode manter facilmente o isolamento do inquilino no OPA ao não armazenar nenhum dado específico do inquilino como um documento base assíncrono. Um documento base assíncrono são dados armazenados na memória e que podem ser atualizados periodicamente, no OPA. Outros

documentos básicos, como a entrada OPA, são transmitidos de forma síncrona e estão disponíveis somente no momento da decisão. Por exemplo, fornecer dados específicos do inquilino como parte da entrada do OPA para uma consulta não constituiria uma violação do isolamento do inquilino, porque esses dados estão disponíveis somente de forma síncrona durante o processo de tomada de decisão.

Integração de inquilinos

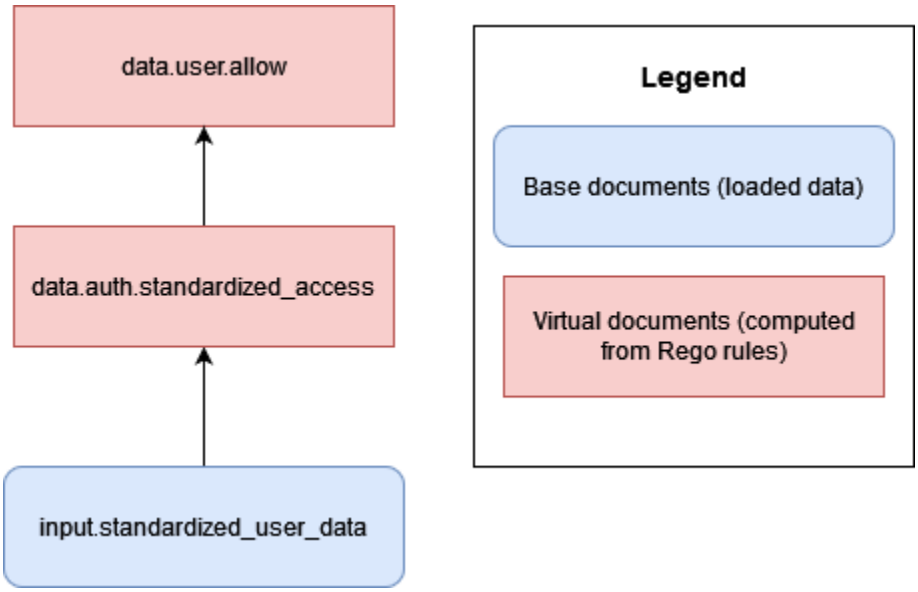
A estrutura dos documentos da OPA deve permitir a integração direta do inquilino sem introduzir requisitos pesados. Você pode organizar documentos virtuais na hierarquia do modelo de documentos OPA com pacotes, e esses pacotes podem conter muitas regras. Ao planejar um modelo de documento OPA para um aplicativo multilocatário, primeiro determine quais dados são necessários para que a OPA tome uma decisão. Você pode fornecer dados como entrada, pré-carregá-los no OPA ou fornecê-los de fontes de dados externas no momento da decisão ou periodicamente. Para obter mais informações sobre o uso de dados externos com o OPA, consulte a seção Recuperação de dados externos para um PDP no OPA, posteriormente neste guia.

Depois de determinar os dados necessários para tomar uma decisão no OPA, considere como implementar as regras do OPA organizadas como pacotes para tomar decisões com esses dados. Por exemplo, em um modelo SaaS em silos em que cada inquilino pode ter requisitos exclusivos sobre como as decisões de autorização são tomadas, você pode implementar pacotes de regras OPA específicos do inquilino.



A desvantagem dessa abordagem é que você deve adicionar um novo conjunto de regras de OPA, específico para cada inquilino, para cada inquilino que você adiciona ao seu aplicativo SaaS. Isso é complicado e difícil de escalar, mas pode ser inevitável, dependendo dos requisitos de seus inquilinos.

Como alternativa, em um modelo SaaS agrupado, se todos os locatários tomarem decisões de autorização com base nas mesmas regras e usarem a mesma estrutura de dados, você poderá usar pacotes OPA padrão que tenham regras geralmente aplicáveis para facilitar a integração de inquilinos e escalar sua implementação de OPA.



Sempre que possível, recomendamos que você use regras e pacotes de OPA generalizados (ou documentos virtuais) para tomar decisões com base em dados padronizados fornecidos por cada inquilino. Essa abordagem torna a OPA facilmente escalável, porque você só altera os dados fornecidos à OPA para cada inquilino, e não a forma como a OPA fornece suas decisões por meio de suas regras. Só é necessário introduzir um rules-per-tenant modelo quando inquilinos individuais exigem decisões exclusivas ou precisam fornecer à OPA dados diferentes dos outros inquilinos.

DevOps, monitorando, registrando e recuperando dados para uma PDP

Nesse paradigma de autorização proposto, as políticas são centralizadas no serviço de autorização. Essa centralização é deliberada porque um dos objetivos dos modelos de design discutidos neste guia é conseguir a dissociação de políticas ou a remoção da lógica de autorização de outros componentes do aplicativo. Tanto o Amazon Verified Permissions quanto o Open Policy Agent (OPA) fornecem mecanismos para atualizar políticas quando mudanças na lógica de autorização são necessárias.

No caso das permissões verificadas, mecanismos para atualização programática de políticas são oferecidos pelo AWS SDK (consulte o Guia de referência da API de permissões verificadas da Amazon). Usando o SDK, você pode promover novas políticas sob demanda. Além disso, como o Verified Permissions é um serviço gerenciado, você não precisa gerenciar, configurar ou manter planos de controle ou agentes para realizar atualizações. No entanto, recomendamos que você use um pipeline de integração contínua e implantação contínua (CI/CD) para administrar a implantação de repositórios de políticas de Permissões Verificadas e atualizações de políticas usando o SDK. AWS

As permissões verificadas fornecem acesso fácil aos recursos de observabilidade. Ele pode ser configurado para registrar todas as tentativas de acesso aos grupos de CloudWatch log da Amazon AWS CloudTrail, aos buckets do Amazon Simple Storage Service (Amazon S3) ou aos fluxos de entrega do Amazon Data Firehose para permitir uma resposta rápida a incidentes de segurança e solicitações de auditoria. Além disso, você pode monitorar a integridade do serviço de Permissões Verificadas por meio do AWS Health Dashboard. Como o Verified Permissions é um serviço gerenciado, sua integridade é mantida por AWS, e você pode configurar recursos de observabilidade usando outros serviços AWS gerenciados.

No caso do OPA, o REST APIs oferece maneiras de atualizar as políticas de forma programática. Você pode configurar o APIs para obter novas versões de pacotes de políticas de locais estabelecidos ou para enviar políticas sob demanda. Além disso, o OPA oferece um serviço básico de descoberta em que novos agentes podem ser configurados dinamicamente e gerenciados centralmente por um plano de controle que distribui pacotes de descoberta. (O plano de controle da OPA deve ser instalado e configurado pelo operador da OPA.) Recomendamos que você crie um CI/CD pipeline robusto para controle de versão, verificação e atualização de políticas, independentemente de o mecanismo de políticas ser Permissões Verificadas, OPA ou outra solução.

Para o OPA, o plano de controle também oferece opções para monitoramento e auditoria. Você pode exportar os registros que contêm as decisões de autorização da OPA para servidores HTTP remotos para agregação de registros. Esses registros de decisão são inestimáveis para fins de auditoria.

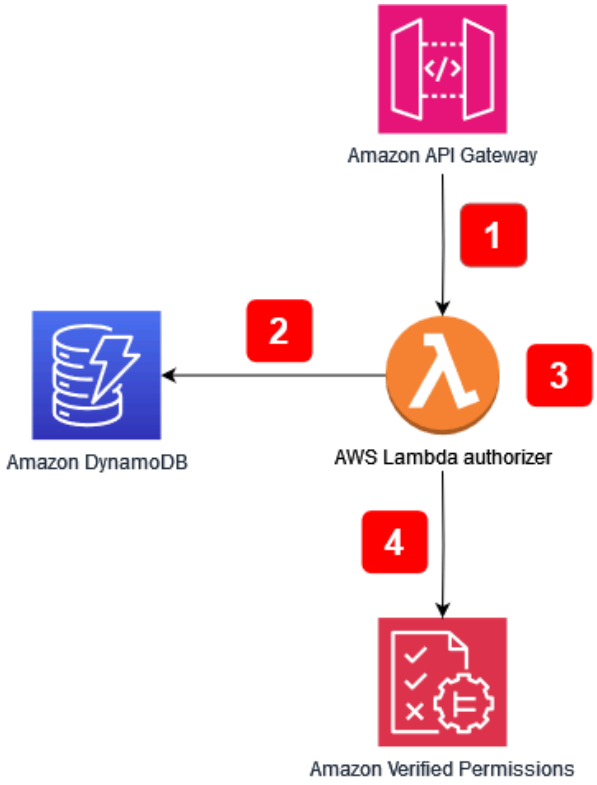
Se você estiver pensando em adotar um modelo de autorização em que as decisões de controle de acesso sejam dissociadas do seu aplicativo, certifique-se de que seu serviço de autorização tenha recursos eficazes de monitoramento, registro e CI/CD gerenciamento para integrar políticas novas PDPs ou atualizá-las.

Tópicos

Recuperação de dados externos para uma PDP nas permissões verificadas da Amazon

O Amazon Verified Permissions não oferece suporte à recuperação de dados externos para uma PDP, mas pode armazenar dados fornecidos pelo usuário como parte de seu esquema. Como na OPA, se todos os dados de uma decisão de autorização puderem ser fornecidos como parte de uma solicitação de autorização ou como parte de um JSON Web Token (JWT) passado como parte da solicitação, nenhuma configuração adicional será necessária. No entanto, você pode fornecer dados adicionais de fontes externas às Permissões Verificadas por meio da solicitação de autorização como parte do serviço autorizador de um aplicativo que chama Permissões Verificadas. Por exemplo, o serviço autorizador de um aplicativo pode consultar dados em uma fonte externa, como o DynamoDB ou o Amazon RDS, e esses serviços podem então incluir os dados fornecidos externamente como parte de uma solicitação de autorização.

O diagrama a seguir mostra um exemplo de como dados adicionais podem ser recuperados e incorporados a uma solicitação de autorização de Permissões Verificadas. Talvez seja necessário usar esse método para recuperar dados, como mapeamentos de funções do RBAC, para recuperar atributos adicionais relevantes aos recursos ou aos principais ou nos casos em que os dados residem em partes diferentes de um aplicativo e não podem ser fornecidos por meio de um token de provedor de identidade (IdP).



Fluxo de aplicação:

1. O aplicativo recebe uma chamada de API para o Amazon API Gateway e encaminha a chamada para o AWS Lambda autorizador.
2. O autorizador do Lambda chama o Amazon DynamoDB para recuperar dados adicionais sobre o principal que fez a solicitação.
3. O autorizador Lambda incorpora os dados adicionais na solicitação de autorização que foi feita às Permissões verificadas.
4. O autorizador Lambda faz uma solicitação de autorização para Permissões verificadas e recebe uma decisão de autorização.

O diagrama inclui um recurso do Amazon API Gateway chamado autorizador Lambda. Embora esse recurso possa não estar disponível para aqueles APIs fornecidos por outros serviços ou aplicativos, você pode replicar o modelo geral de uso de um autorizador para buscar dados adicionais para incorporar a uma solicitação de autorização de Permissões Verificadas em vários casos de uso.

Recuperando dados externos para um PDP no OPA

Para o OPA, se todos os dados necessários para uma decisão de autorização puderem ser fornecidos como entrada ou como parte de um JSON Web Token (JWT) passado como componente da consulta, nenhuma configuração adicional será necessária. (É relativamente simples passar JWTs dados de contexto SaaS para o OPA como parte da entrada da consulta.) O OPA pode aceitar entradas JSON arbitrárias no que é chamado de abordagem de entrada de sobrecarga. Se um PDP exigir dados além do que pode ser incluído como entrada ou JWT, o OPA fornece várias opções para recuperar esses dados. Isso inclui agrupamento, envio de dados (replicação) e recuperação dinâmica de dados.

Pacote OPA

O recurso de agrupamento OPA suporta o seguinte processo para recuperação externa de dados:

1. O ponto de fiscalização da política (PEP) solicita uma decisão de autorização.
2. A OPA baixa novos pacotes de políticas, incluindo dados externos.
3. O serviço de empacotamento replica dados da (s) fonte (s) de dados.

Quando você usa o recurso de agrupamento, o OPA baixa periodicamente pacotes de políticas e dados de um serviço de pacote centralizado. (A OPA não fornece a implementação e a configuração de um serviço de pacote.) Todas as políticas e dados externos extraídos do serviço de pacote são armazenados na memória. Essa opção não funcionará se o tamanho dos dados externos for muito grande para ser armazenado na memória ou se os dados mudarem com muita frequência.

Para obter mais informações sobre o recurso de agrupamento, consulte a documentação do OPA.

Replicação de OPA (envio de dados)

A abordagem de replicação OPA suporta o seguinte processo de recuperação de dados externos:

1. O PEP solicita uma decisão de autorização.
2. O replicador de dados envia os dados para o OPA.
3. O replicador de dados replica os dados da (s) fonte (s) de dados.

Nessa alternativa à abordagem de agrupamento, os dados são enviados para, em vez de serem retirados periodicamente pela OPA. (O OPA não fornece a implementação e a configuração de um

replicador.) A abordagem push tem as mesmas limitações de tamanho de dados que a abordagem de agrupamento, porque o OPA armazena todos os dados na memória. A principal vantagem da opção push é que você pode atualizar os dados no OPA por deltas em vez de substituir todos os dados externos a cada vez. Isso torna a opção push mais apropriada para conjuntos de dados que mudam com frequência.

Para obter mais informações sobre a opção de replicação, consulte a documentação do OPA.

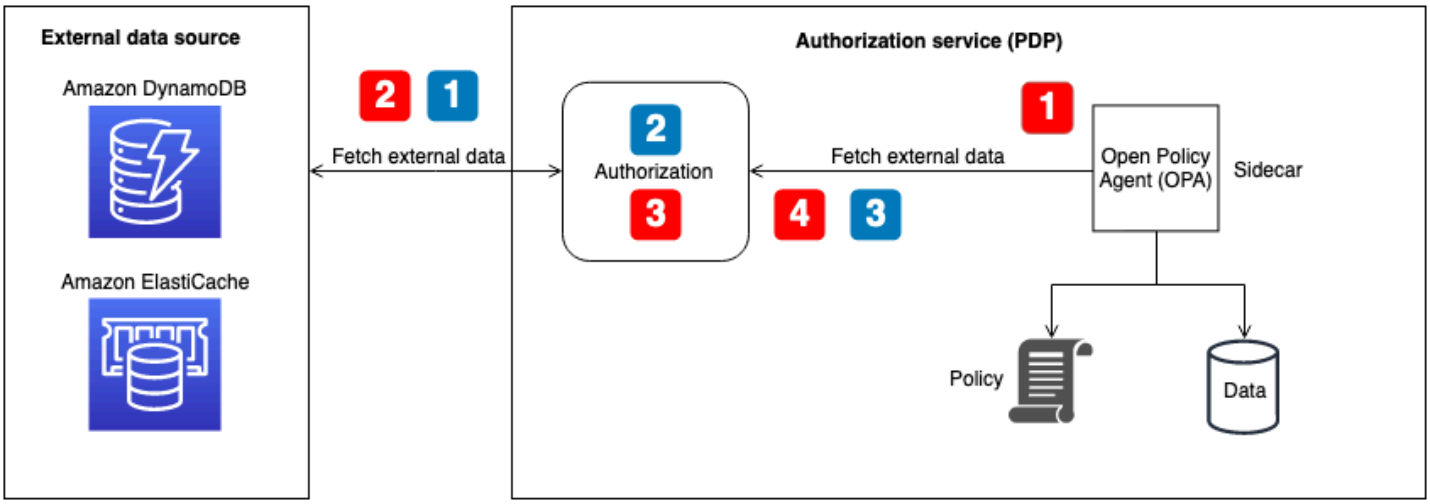
Recuperação dinâmica de dados OPA

Se os dados externos a serem recuperados forem muito grandes para serem armazenados em cache na memória do OPA, os dados poderão ser extraídos dinamicamente de uma fonte externa durante a avaliação de uma decisão de autorização. Quando você usa essa abordagem, os dados estão sempre atualizados. Essa abordagem tem duas desvantagens: latência e acessibilidade da rede. Atualmente, o OPA pode recuperar dados em tempo de execução somente por meio de uma solicitação HTTP. Se as chamadas que vão para uma fonte de dados externa não puderem retornar dados como uma resposta HTTP, elas precisarão de uma API personalizada ou de algum outro mecanismo para fornecer esses dados à OPA. Como o OPA pode recuperar dados somente por meio de solicitações HTTP, e a velocidade de recuperação dos dados é fundamental, recomendamos que você use um como o AWS service (Serviço da AWS) Amazon DynamoDB para armazenar dados externos sempre que possível.

Para obter mais informações sobre a abordagem pull, consulte a documentação do OPA.

Usando um serviço de autorização para implementação com OPA

Quando você busca dados externos usando agrupamento, replicação ou uma abordagem de extração dinâmica, recomendamos que o serviço de autorização facilite essa interação. Isso ocorre porque o serviço de autorização pode recuperar dados externos e transformá-los em JSON para que o OPA tome decisões de autorização. O diagrama a seguir mostra como um serviço de autorização pode funcionar com essas três abordagens externas de recuperação de dados.



Recuperação de dados externos para fluxo de OPA — pacote ou recuperação dinâmica de dados no momento da decisão (ilustrada com textos explicativos numerados em vermelho no diagrama):

1. O OPA chama o endpoint local da API para o serviço de autorização, que é configurado como um endpoint de pacote ou o endpoint para recuperação dinâmica de dados durante as decisões de autorização.
2. O serviço de autorização consulta ou chama a fonte de dados externa para recuperar dados externos. (Para um endpoint de pacote, esses dados também devem conter políticas e regras de OPA. As atualizações do pacote substituem tudo, tanto os dados quanto as políticas, no cache do OPA.)
3. O serviço de autorização realiza qualquer transformação necessária nos dados retornados para transformá-los na entrada JSON esperada.
4. Os dados são devolvidos à OPA. Ele é armazenado em cache na memória para configuração do pacote e usado imediatamente para decisões dinâmicas de autorização.

Recuperação de dados externos para o fluxo OPA — replicador (ilustrado com textos explicativos com números azuis no diagrama):

1. O replicador (parte do serviço de autorização) chama a fonte de dados externa e recupera todos os dados a serem atualizados no OPA. Isso pode incluir políticas, regras e dados externos. Essa chamada pode estar em uma cadência definida ou pode ocorrer em resposta a atualizações de dados na fonte externa.
2. O serviço de autorização realiza todas as transformações necessárias nos dados retornados para transformá-los na entrada JSON esperada.

3. O serviço de autorização chama o OPA e armazena em cache os dados na memória. O serviço de autorização pode atualizar dados, políticas e regras de forma seletiva.

Recomendações para isolamento de inquilinos e privacidade de dados

A seção anterior forneceu várias abordagens para o uso de dados externos com OPA e Amazon Verified Permissions para auxiliar na tomada de decisões de autorização. Sempre que possível, recomendamos que você use a abordagem de entrada de sobrecarga para passar dados de contexto SaaS para o OPA para tomar decisões de autorização em vez de armazenar dados na memória do OPA. Esse caso de uso não se aplica a AWS Cloud Map, porque não oferece suporte ao armazenamento de dados externos no serviço.

Nos modelos híbridos de controle de acesso baseado em função (RBAC) ou RBAC e controle de acesso baseado em atributos (ABAC), os dados fornecidos somente por uma solicitação ou consulta de autorização podem ser insuficientes, pois as funções e permissões precisam ser referenciadas para tomar decisões de autorização. Para manter o isolamento do inquilino e a privacidade do mapeamento de funções, esses dados não devem residir na OPA. Os dados do RBAC devem residir em uma fonte de dados externa, como um banco de dados, ou devem ser passados como parte das declarações de um IdP em um JWT. Nas Permissões verificadas, os dados do RBAC podem ser mantidos como parte das políticas e do esquema no modelo de armazenamento de políticas por inquilino, porque cada inquilino tem seu próprio armazenamento de políticas separado logicamente. No entanto, em um modelo de armazenamento de políticas compartilhado para vários locatários, os dados de mapeamento de funções não devem residir nas Permissões Verificadas para manter o isolamento do inquilino.

Além disso, a OPA e as Permissões Verificadas não devem ser usadas para mapear funções predefinidas para permissões específicas, pois isso dificulta que os inquilinos definam suas próprias funções e permissões. Isso também torna sua lógica de autorização rígida e precisa de atualização constante. A exceção a essa diretriz é o modelo de armazenamento de políticas por inquilino em Permissões verificadas, porque esse modelo permite que cada inquilino tenha suas próprias políticas que podem ser avaliadas de forma independente por inquilino.

Amazon Verified Permissions

O único lugar em que as Permissões Verificadas podem armazenar dados RBAC potencialmente privados é no esquema. Isso é aceitável no modelo de armazenamento de políticas por inquilino,

porque cada inquilino tem seu próprio repositório de políticas separado logicamente. No entanto, isso pode comprometer o isolamento de inquilinos em um modelo compartilhado de armazenamento de políticas multilocatário. Nos casos em que esses dados são necessários para tomar uma decisão de autorização, eles devem ser recuperados de uma fonte externa, como o DynamoDB ou o Amazon RDS, e incorporados à solicitação de autorização de permissões verificadas.

OPA

As abordagens seguras com OPA para manter a privacidade e o isolamento dos inquilinos dos dados do RBAC incluem o uso de recuperação ou replicação dinâmica de dados para obter dados externos. Isso ocorre porque você pode usar o serviço de autorização ilustrado no diagrama anterior para fornecer somente dados externos específicos do inquilino ou do usuário para tomar uma decisão de autorização. Por exemplo, você pode usar um replicador para fornecer dados RBAC ou uma matriz de permissões para o cache OPA quando um usuário faz login e fazer com que os dados sejam referenciados com base em um usuário fornecido nos dados de entrada. Você pode usar uma abordagem semelhante com dados extraídos dinamicamente para recuperar somente os dados relevantes para tomar decisões de autorização. Além disso, na abordagem dinâmica de recuperação de dados, esses dados não precisam ser armazenados em cache no OPA. A abordagem de agrupamento não é tão eficaz quanto a abordagem de recuperação dinâmica para manter o isolamento do inquilino, porque ela atualiza tudo no cache do OPA e não pode processar atualizações precisas. O modelo de agrupamento ainda é uma boa abordagem para atualizar políticas de OPA e dados não RBAC.

Práticas recomendadas

Esta seção lista algumas das principais conclusões deste guia. Para discussões detalhadas sobre cada ponto, siga os links para as seções correspondentes.

Selecione um modelo de controle de acesso que funcione para seu aplicativo

Este guia aborda vários modelos de controle de acesso. Dependendo do aplicativo e dos requisitos comerciais, você deve selecionar um modelo que funcione para você. Considere como você pode usar esses modelos para atender às suas necessidades de controle de acesso e como suas necessidades de controle de acesso podem evoluir, exigindo mudanças na abordagem selecionada.

Implemente um PDP

O ponto de decisão política (PDP) pode ser caracterizado como um mecanismo de políticas ou regras. Esse componente é responsável por aplicar políticas ou regras e retornar uma decisão sobre se um determinado acesso é permitido. Um PDP permite que a lógica de autorização no código do aplicativo seja transferida para um sistema separado. Isso pode simplificar o código do aplicativo. Ele também fornece uma interface easy-to-use idempotente para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou qualquer outro componente do aplicativo. Um PDP pode ser usado para impor os requisitos de locação de forma consistente em um aplicativo.

Implemente PEPs para cada API em seu aplicativo

A implementação de um ponto de fiscalização de políticas (PEP) exige determinar onde a fiscalização do controle de acesso deve ocorrer em um aplicativo. Como primeira etapa, localize os pontos em seu aplicativo onde você pode incorporar PEPs. Considere esse princípio ao decidir onde adicionar PEPs:

Se um aplicativo expõe uma API, deve haver autorização e controle de acesso nessa API.

Considere usar Amazon Verified Permissions ou OPA como um mecanismo de política para seu PDP

As permissões verificadas da Amazon têm vantagens em relação aos mecanismos de política personalizados. É um serviço de autorização e gerenciamento de permissões escalável e refinado para os aplicativos que você cria. Ele suporta políticas de escrita na linguagem declarativa de código aberto de alto nível Cedar. Como resultado, implementar um mecanismo de políticas usando permissões verificadas exige menos esforço de desenvolvimento do que implementar sua própria solução. Além disso, as Permissões Verificadas são totalmente gerenciadas, para que você não precise gerenciar a infraestrutura subjacente.

O Open Policy Agent (OPA) tem vantagens sobre os mecanismos de política personalizados. A OPA e sua avaliação de políticas com a Rego fornecem um mecanismo de políticas flexível e pré-construído que suporta a redação de políticas em uma linguagem declarativa de alto nível. Isso faz com que o nível de esforço necessário para implementar um mecanismo de políticas seja significativamente menor do que para criar sua própria solução. Além disso, o OPA está rapidamente se tornando um padrão de autorização bem suportado.

Implemente um plano de controle para DevOps OPA para monitoramento e registro

Como o OPA não fornece um meio de atualizar e rastrear alterações na lógica de autorização por meio do controle de origem, recomendamos que você implemente um plano de controle para executar essas funções. Isso permitirá que as atualizações sejam distribuídas mais facilmente aos agentes do OPA, especialmente se o OPA estiver operando em um sistema distribuído, o que reduzirá a carga administrativa do uso do OPA. Além disso, um plano de controle pode ser usado para coletar registros para agregação e monitorar o status dos agentes OPA.

Configure os recursos de registro e observabilidade nas Permissões verificadas

As permissões verificadas fornecem acesso fácil aos recursos de observabilidade. Você pode configurar o serviço para registrar todas as tentativas de acesso a grupos de CloudWatch registros da Amazon AWS CloudTrail, buckets S3 ou fluxos de entrega do Amazon Data Firehose para permitir uma resposta rápida a incidentes de segurança e solicitações de auditoria. Além disso,

você pode monitorar a integridade do serviço por meio do AWS Health Dashboard. Como o Verified Permissions é um serviço gerenciado, sua integridade é mantida por AWS, e você pode configurar seus recursos de observabilidade usando outros serviços AWS gerenciados.

Use um CI/CD pipeline para provisionar e atualizar repositórios e políticas de políticas em Permissões verificadas

O Verified Permissions é um serviço gerenciado, portanto, você não precisa gerenciar, configurar ou manter planos de controle ou agentes para realizar atualizações. No entanto, ainda recomendamos que você use um pipeline de integração contínua e implantação contínua (CI/CD) para administrar a implantação de repositórios de políticas de Permissões Verificadas e atualizações de políticas usando o SDK. AWS Esse esforço pode eliminar o esforço manual e reduzir a probabilidade de erros do operador ao fazer alterações nos recursos de permissões verificadas.

Determine se os dados externos são necessários para decisões de autorização e selecione um modelo para acomodá-los

Se um PDP puder tomar decisões de autorização com base apenas nos dados contidos em um JSON Web Token (JWT), geralmente não é necessário importar dados externos para auxiliar na tomada dessas decisões. Se você usa Permissões verificadas ou OPA como PDP, ele também pode aceitar entradas adicionais passadas como parte da solicitação, mesmo que esses dados não estejam incluídos em um JWT. Para Permissões verificadas, você pode usar um parâmetro de contexto para os dados adicionais. Para OPA, você pode usar dados JSON como entrada de sobrecarga. Se você usa um JWT, os métodos de entrada de contexto ou sobrecarga geralmente são muito mais fáceis do que manter dados externos em outra fonte. Se forem necessários dados externos mais complexos para tomar decisões de autorização, a OPA oferece vários modelos para recuperar dados externos, e as Permissões Verificadas podem complementar os dados em suas solicitações de autorização referenciando fontes externas com um serviço de autorização.

Perguntas frequentes

Esta seção fornece respostas às perguntas mais comuns sobre a implementação do controle de acesso e autorização da API em aplicativos SaaS multilocatários.

P: Qual é a diferença entre autorização e autenticação?

R. A autenticação é o processo de verificar quem é um usuário. A autorização concede permissões aos usuários para acessar um recurso específico.

P: Qual é a diferença entre autorização e isolamento de inquilinos em um aplicativo SaaS?

R. O isolamento de inquilinos se refere a mecanismos explícitos usados em um sistema SaaS para garantir que os recursos de cada inquilino, mesmo quando operando em infraestrutura compartilhada, sejam isolados. A autorização multilocatária se refere à autorização de ações de entrada e à prevenção de que elas sejam implementadas no inquilino errado. Um usuário hipotético pode ser autenticado e autorizado, mas ainda pode acessar os recursos de outro inquilino. Nada sobre autenticação e autorização necessariamente bloqueia esse acesso, mas o isolamento do inquilino é necessário para atingir esse objetivo. Para obter mais informações sobre esses dois conceitos, consulte a discussão sobre isolamento de inquilinos no whitepaper AWS SaaS Architecture Fundamentals.

P: Por que preciso considerar o isolamento de inquilinos para meu aplicativo SaaS?

R. Os aplicativos SaaS têm vários inquilinos. Um inquilino pode ser uma organização cliente ou qualquer entidade externa que use esse aplicativo SaaS. Dependendo de como o aplicativo foi projetado, isso significa que os inquilinos podem estar acessando bancos de dados compartilhados APIs ou outros recursos. É importante manter o isolamento do inquilino, ou seja, construções que controlem rigorosamente o acesso aos recursos e bloqueiem qualquer tentativa de acessar os recursos de outro inquilino, para impedir que os usuários de um inquilino acessem as informações privadas de outro inquilino. Os aplicativos SaaS geralmente são projetados para garantir que o isolamento do inquilino seja mantido em todo o aplicativo e que os inquilinos possam acessar somente seus próprios recursos.

P: Por que preciso de um modelo de controle de acesso?

R. Os modelos de controle de acesso são usados para criar um método consistente de determinar como conceder acesso aos recursos em um aplicativo. Isso pode ser feito atribuindo funções a usuários que estejam estreitamente alinhadas com a lógica de negócios, ou pode ser baseado

em outros atributos contextuais, como a hora do dia ou se um usuário atende a uma condição predefinida. Os modelos de controle de acesso formam a lógica básica que seu aplicativo usa ao tomar decisões de autorização para determinar as permissões do usuário.

P: O controle de acesso à API é necessário para meu aplicativo?

R. Sim. APIs deve sempre verificar se o chamador tem o acesso apropriado. O controle de acesso generalizado à API também garante que o acesso seja concedido apenas com base nos inquilinos, para que o isolamento apropriado possa ser mantido.

P: Por que os mecanismos de políticas são PDPs recomendados para autorização?

R. Um ponto de decisão de política (PDP) permite que a lógica de autorização no código do aplicativo seja transferida para um sistema separado. Isso pode simplificar o código do aplicativo. Ele também fornece uma interface easy-to-use idempotente para tomar decisões de autorização para microsserviços APIs, camadas de Backend for Frontend (BFF) ou qualquer outro componente do aplicativo.

P: O que é um PEP?

R. Um ponto de fiscalização de políticas (PEP) é responsável por receber solicitações de autorização que são enviadas ao PDP para avaliação. Um PEP pode estar em qualquer lugar em um aplicativo em que os dados e os recursos devem ser protegidos ou onde a lógica de autorização é aplicada. PEPs são relativamente simples em comparação com PDPs. Um PEP é responsável somente por solicitar e avaliar uma decisão de autorização e não exige que nenhuma lógica de autorização seja incorporada a ela.

P: Como devo escolher entre Amazon Verified Permissions e OPA?

R. Para escolher entre permissões verificadas e Open Policy Agent (OPA), tenha sempre em mente seu caso de uso e seus requisitos exclusivos. As Permissões Verificadas fornecem uma maneira totalmente gerenciada de definir permissões refinadas, auditar permissões em todos os aplicativos e centralizar o sistema de administração de políticas para seus aplicativos, ao mesmo tempo em que atende aos requisitos de latência do aplicativo com processamento em milissegundos. O OPA é um mecanismo de políticas de uso geral de código aberto que também pode ajudá-lo a unificar a política em toda a sua pilha de aplicativos. Para executar o OPA, você precisa hospedá-lo em seu AWS ambiente, normalmente com um contêiner ou AWS Lambda funções.

O Verified Permissions usa a linguagem de política de código aberto Cedar, enquanto o OPA usa o Rego. Portanto, a familiaridade com um desses idiomas pode fazer com que você escolha essa

solução. No entanto, recomendamos que você leia sobre as duas linguagens e depois resolva o problema que está tentando resolver para encontrar a melhor solução para seu caso de uso.

P: Existem alternativas de código aberto às permissões verificadas e ao OPA?

R. Existem alguns sistemas de código aberto que são semelhantes às Permissões Verificadas e ao Open Policy Agent (OPA), como a Common Expression Language (CEL). Este guia se concentra tanto nas Permissões Verificadas, como um serviço de gerenciamento de permissões escalável e de autorização refinado, quanto na OPA, que é amplamente adotada, documentada e adaptável a diversos tipos de aplicativos e requisitos de autorização.

P: Preciso escrever um serviço de autorização para usar o OPA ou posso interagir diretamente com o OPA?

R. Você pode interagir diretamente com a OPA. Um serviço de autorização no contexto desta orientação se refere a um serviço que traduz solicitações de decisão de autorização em consultas OPA e vice-versa. Se seu aplicativo puder consultar e aceitar respostas de OPA diretamente, não há necessidade de introduzir essa complexidade adicional.

P: Como faço para monitorar meus agentes de OPA para fins de disponibilidade e auditoria?

R. O OPA fornece registro e monitoramento básico do tempo de atividade, embora sua configuração padrão provavelmente seja insuficiente para implantações corporativas. Para obter mais informações, consulte a DevOps seção Monitoramento e registro anterior neste guia.

P: Como posso monitorar as permissões verificadas para fins de disponibilidade e auditoria?

R. O Verified Permissions é um serviço AWS gerenciado e pode ser monitorado quanto à disponibilidade por meio do AWS Health Dashboard. Além disso, o Verified Permissions é capaz de fazer AWS CloudTrail login no Amazon CloudWatch Logs, no Amazon S3 e no Amazon Data Firehose.

P: Quais sistemas operacionais e AWS serviços posso usar para executar o OPA?

R. Você pode executar o OPA no macOS, Windows e Linux. Os agentes OPA podem ser configurados em agentes do Amazon Elastic Compute Cloud EC2 (Amazon), bem como em serviços de conteinerização, como o Amazon Elastic Container Service (Amazon ECS) e o Amazon Elastic Kubernetes Service (Amazon EKS).

P: Quais sistemas operacionais e AWS serviços posso usar para executar as Permissões Verificadas?

A. O Verified Permissions é um serviço AWS gerenciado e é operado pela AWS. Nenhuma configuração, instalação ou hospedagem adicional é necessária para usar as Permissões Verificadas, exceto pela capacidade de fazer solicitações de autorização para o serviço.

P: Posso executar o OPA em? AWS Lambda

R. Você pode executar o OPA na biblioteca Lambda as a Go. Para obter informações sobre como você pode fazer isso com um autorizador Lambda do API Gateway, consulte a postagem do AWS blog Criando um autorizador Lambda personalizado usando o Open Policy Agent.

P: Como devo decidir entre uma abordagem de PDP distribuída e uma abordagem de PDP centralizada?

R. Isso depende da sua aplicação. Provavelmente, será determinado com base na diferença de latência entre um modelo PDP distribuído e centralizado. Recomendamos que você crie uma prova de conceito e teste o desempenho do seu aplicativo para verificar sua solução.

P: Além disso, posso usar o OPA para casos de uso? APIs

R. Sim. A documentação do OPA fornece exemplos para Kubernetes, Envoy, Docker, Kafka, SSH e sudo e Terraform. Além disso, o OPA pode retornar respostas JSON arbitrárias às consultas usando regras parciais do Rego. Dependendo da consulta, o OPA pode ser usado para responder a muitas perguntas com respostas JSON.

P: Além disso APIs, posso usar as permissões verificadas para casos de uso?

R. Sim. As permissões verificadas podem fornecer uma `DENY` resposta `ALLOW` ou para qualquer solicitação de autorização recebida. As permissões verificadas podem fornecer respostas de autorização para qualquer aplicativo ou serviço que exija decisões de autorização.

P: Posso criar políticas em Permissões verificadas usando a linguagem de política do IAM?

R. Não. Você deve usar a linguagem de política do Cedar para criar políticas. O Cedar foi projetado para oferecer suporte ao gerenciamento de permissões para recursos de aplicativos do cliente, enquanto a linguagem de política AWS Identity and Access Management (IAM) evoluiu para oferecer suporte ao controle de acesso aos AWS recursos.

Próximas etapas

É possível superar a complexidade da autorização e do controle de acesso à API para aplicações SaaS multilocatário com a adoção de uma abordagem padronizada e independente de linguagem para a tomada de decisões de autorização. Essas abordagens incorporam pontos de decisão política (PDPs) e pontos de aplicação de políticas (PDPs) que impõem o acesso de maneira flexível e abrangente. Várias abordagens de controle de acesso, como controle de acesso por perfil (RBAC), controle de acesso por atributo (ABAC) ou uma combinação dos dois, podem ser incorporadas a uma estratégia coesa de controle de acesso. A remoção da lógica de autorização de uma aplicação elimina a sobrecarga de incluir soluções ad hoc no código da aplicação para abordar o controle de acesso. A implementação e as práticas recomendadas discutidas neste guia têm como objetivo informar e padronizar uma abordagem para a implementação da autorização e do controle de acesso à API em aplicações SaaS multilocatário. Você pode usar essa orientação como a primeira etapa na coleta de informações e na criação de um sistema robusto de controle de acesso e autorização para sua aplicação. Próximas etapas:

- Analise suas necessidades de autorização e isolamento de locatários e selecione um modelo de controle de acesso para sua aplicação.
- Crie uma prova de conceito para testes usando o Amazon Verified Permissions ou o Open Policy Agent (OPA), ou escrevendo seu próprio mecanismo de política personalizado.
- Identifique APIs e localize em seu aplicativo onde PEPs deve ser implementado.

Recursos

Referências

- Documentação de permissões verificadas da Amazon (AWS documentação)
- Como usar as permissões verificadas da Amazon para autorização (postagem AWS no blog)
- Implemente um provedor de política de autorização personalizado para aplicativos ASP.NET Core usando permissões verificadas da Amazon (AWS postagem do blog)
- Gerencie funções e direitos com o PBAC usando as permissões AWS verificadas da Amazon (postagem no blog)
- Controle de acesso SaaS usando Amazon Verified Permissions com um armazenamento de políticas por inquilino (postagem no blog)AWS
- A documentação oficial da OPA
- Por que as empresas devem adotar o projeto CNCF mais recentemente graduado — Open Policy Agent (artigo da Forbes de Janakiram MSV, 8 de fevereiro de 2021)
- Criação de um autorizador Lambda personalizado usando o Open Policy Agent (AWS postagem no blog)
- Realize a política como código com o AWS Cloud Development Kit por meio do Open Policy Agent (postagem AWS no blog)
- Governança da nuvem e conformidade AWS com a política como código (postagem AWS no blog)
- Usando o Open Policy Agent no Amazon EKS (publicação AWS do blog)
- Conformidade como código para o Amazon ECS usando o Open Policy Agent, a Amazon EventBridge e AWS Lambda (postagem AWS no blog)
- Contramedidas baseadas em políticas para Kubernetes — Parte 1 (postagem no blog)AWS
- Usando autorizadores Lambda do API Gateway (documentação)AWS

Ferramentas

- The Cedar Playground (para testar o Cedar em um navegador)
- Repositório Cedar Github
- Referência da linguagem Cedar
- The Rego Playground (para testar o Rego em um navegador)

- Repositório OPA GitHub

Parceiros

- Parceiros de gerenciamento de identidade e acesso
- Parceiros de segurança de aplicativos
- Parceiros de governança de nuvem
- Parceiros de segurança e conformidade
- Parceiros de operações de segurança e automação
- Parceiros de engenharia de segurança

Histórico do documento

A tabela a seguir descreve alterações significativas feitas neste guia. Se desejar receber notificações sobre futuras atualizações, inscreva-se em um feed RSS.

Alteração	Descrição	Data
Detalhes e exemplos adicionados para Amazon Verified Permissions	Foram adicionadas discussões detalhadas, exemplos e códigos para usar o Amazon Verified Permissions para implementar uma PDP. As novas seções incluem: • Implementando uma PDP usando as permissões verificadas da Amazon • Modelos de design para Amazon Verified Permissions • Considerações sobre o design multilocatário do Amazon Verified Permissions • Recuperação de dados externos para uma PDP nas permissões verificadas da Amazon	28 de maio de 2024
Informações esclarecidas	Esclareceu o PDP distribuído com um APIs modelo PEPs de design.	10 de janeiro de 2024
Foram adicionadas informações breves sobre o novo AWS serviço	Foram adicionadas informações sobre as Permissões Verificadas da Amazon,	22 de maio de 2023

	que oferecem as mesmas funcionalidades e benefícios da OPA.	
—	Publicação inicial	17 de agosto de 2021

AWS Glossário de orientação prescritiva

A seguir estão os termos comumente usados em estratégias, guias e padrões fornecidos pela Orientação AWS Prescritiva. Para sugerir entradas, use o link Fornecer feedback no final do glossário.

Números

7 Rs

Sete estratégias comuns de migração para mover aplicações para a nuvem. Essas estratégias baseiam-se nos 5 Rs identificados pela Gartner em 2011 e consistem em:

- Refatorar/rearquitetar: mova uma aplicação e modifique sua arquitetura aproveitando ao máximo os recursos nativos de nuvem para melhorar a agilidade, a performance e a escalabilidade. Isso normalmente envolve a portabilidade do sistema operacional e do banco de dados. Exemplo: migrar seu banco de dados Oracle on-premises para o Amazon Aurora Edição Compatível com PostgreSQL.
- Redefinir a plataforma (mover e redefinir [mover e redefinir (lift-and-reshape]): mova uma aplicação para a nuvem e introduza algum nível de otimização a fim de aproveitar os recursos da nuvem. Exemplo: migrar seu banco de dados Oracle on-premises para o Amazon Relational Database Service (Amazon RDS) para Oracle na Nuvem AWS.
- Recomprar (drop and shop): mude para um produto diferente, normalmente migrando de uma licença tradicional para um modelo SaaS. Exemplo: migrar seu sistema de gerenciamento de relacionamento com o cliente (CRM) para o Salesforce.com.
- Redefinir a hospedagem (mover sem alterações [lift-and-shift])mover uma aplicação para a nuvem sem fazer nenhuma alteração a fim de aproveitar os recursos da nuvem. Exemplo: migrar seu banco de dados Oracle on-premises para o Oracle em uma instância do EC2 na Nuvem AWS.
- Realocar (mover o hipervisor sem alterações [hypervisor-level lift-and-shift]): mover a infraestrutura para a nuvem sem comprar novo hardware, reescrever aplicações ou modificar suas operações existentes. Você migra servidores de uma plataforma on-premises para um serviço de nuvem para a mesma plataforma. Exemplo: Migrar um Microsoft Hyper-V aplicativo para o. AWS
- Reter (revisitar): mantenha as aplicações em seu ambiente de origem. Isso pode incluir aplicações que exigem grande refatoração, e você deseja adiar esse trabalho para um

momento posterior, e aplicações antigas que você deseja manter porque não há justificativa comercial para migrá-las.

- Retirar: desative ou remova aplicações que não são mais necessárias em seu ambiente de origem.

A

ABAC

Consulte controle de acesso baseado em atributo.

serviços abstraídos

Veja serviços gerenciados.

ACID

Veja atomicidade, consistência, isolamento, durabilidade.

migração ativa-ativa

Um método de migração de banco de dados no qual os bancos de dados de origem e de destino são mantidos em sincronia (por meio de uma ferramenta de replicação bidirecional ou operações de gravação dupla), e ambos os bancos de dados lidam com transações de aplicações conectadas durante a migração. Esse método oferece suporte à migração em lotes pequenos e controlados, em vez de exigir uma substituição única. É mais flexível, mas exige mais trabalho do que a migração ativa-passiva.

migração ativa-passiva

Um método de migração de banco de dados em que os bancos de dados de origem e de destino são mantidos em sincronia, mas somente o banco de dados de origem manipula as transações das aplicações conectadas, enquanto os dados são replicados no banco de dados de destino. O banco de dados de destino não aceita nenhuma transação durante a migração.

AGGREGATE FUNCTION

Uma função SQL que opera em um grupo de linhas e calcula um único valor de retorno para o grupo. Exemplos de funções agregadas incluem `SUM` e `MAX`.

AI

Veja inteligência artificial.

AIOps

Veja operações de inteligência artificial.

anonimização

O processo de excluir permanentemente informações pessoais em um conjunto de dados. A anonimização pode ajudar a proteger a privacidade pessoal. Dados anônimos não são mais considerados dados pessoais.

antipadrões

Uma solução frequentemente usada para um problema recorrente em que a solução é contraproducente, ineficaz ou menos eficaz do que uma alternativa.

controle de aplicações

Uma abordagem de segurança que permite o uso somente de aplicações aprovadas para ajudar a proteger um sistema contra malware.

portfólio de aplicações

Uma coleção de informações detalhadas sobre cada aplicação usada por uma organização, incluindo o custo para criar e manter a aplicação e seu valor comercial. Essas informações são fundamentais para o processo de descoberta e análise de portfólio e ajudam a identificar e priorizar as aplicações a serem migradas, modernizadas e otimizadas.

inteligência artificial (IA)

O campo da ciência da computação que se dedica ao uso de tecnologias de computação para desempenhar funções cognitivas normalmente associadas aos humanos, como aprender, resolver problemas e reconhecer padrões. Para obter mais informações, consulte O que é inteligência artificial?

operações de inteligência artificial (AIOps)

O processo de usar técnicas de machine learning para resolver problemas operacionais, reduzir incidentes operacionais e intervenção humana e aumentar a qualidade do serviço. Para obter mais informações sobre como AIOps é usado na estratégia de AWS migração, consulte o guia de integração de operações.

criptografia assimétrica

Um algoritmo de criptografia que usa um par de chaves, uma chave pública para criptografia e uma chave privada para descriptografia. É possível compartilhar a chave pública porque ela não é usada na descriptografia, mas o acesso à chave privada deve ser altamente restrito.

atomicidade, consistência, isolamento, durabilidade (ACID)

Um conjunto de propriedades de software que garantem a validade dos dados e a confiabilidade operacional de um banco de dados, mesmo no caso de erros, falhas de energia ou outros problemas.

controle de acesso por atributo (ABAC)

A prática de criar permissões minuciosas com base nos atributos do usuário, como departamento, cargo e nome da equipe. Para obter mais informações, consulte ABAC AWS na documentação AWS Identity and Access Management (IAM).

fonte de dados autorizada

Um local onde você armazena a versão principal dos dados, que é considerada a fonte de informações mais confiável. Você pode copiar dados da fonte de dados autorizada para outros locais com o objetivo de processar ou modificar os dados, como anonimizá-los, redigi-los ou pseudonimizá-los.

Zona de disponibilidade

Um local distinto dentro de um Região da AWS que está isolado de falhas em outras zonas de disponibilidade e fornece conectividade de rede barata e de baixa latência a outras zonas de disponibilidade na mesma região.

AWS Estrutura de adoção da nuvem (AWS CAF)

Uma estrutura de diretrizes e melhores práticas AWS para ajudar as organizações a desenvolver um plano eficiente e eficaz para migrar com sucesso para a nuvem. AWS O CAF organiza a orientação em seis áreas de foco chamadas perspectivas: negócios, pessoas, governança, plataforma, segurança e operações. As perspectivas de negócios, pessoas e governança têm como foco habilidades e processos de negócios; as perspectivas de plataforma, segurança e operações concentram-se em habilidades e processos técnicos. Por exemplo, a perspectiva das pessoas tem como alvo as partes interessadas que lidam com recursos humanos (RH), funções de pessoal e gerenciamento de pessoal. Nessa perspectiva, o AWS CAF fornece orientação para desenvolvimento, treinamento e comunicação de pessoas para ajudar a preparar a organização

para a adoção bem-sucedida da nuvem. Para obter mais informações, consulte o site da AWS CAF e o whitepaper da AWS CAF.

AWS Estrutura de qualificação da carga de trabalho (AWS WQF)

Uma ferramenta que avalia as cargas de trabalho de migração do banco de dados, recomenda estratégias de migração e fornece estimativas de trabalho. AWS O WQF está incluído com AWS Schema Conversion Tool ()AWS SCT. Ela analisa esquemas de banco de dados e objetos de código, código de aplicações, dependências e características de performance, além de fornecer relatórios de avaliação.

B

bot malicioso

Um bot destinado a causar disrupção ou danos a indivíduos ou organizações.

BCP

Veja planejamento de continuidade de negócios

gráfico de comportamento

Uma visualização unificada e interativa do comportamento e das interações de recursos ao longo do tempo. É possível usar um gráfico de comportamento com o Amazon Detective para examinar tentativas de login malsucedidas, chamadas de API suspeitas e ações similares. Para obter mais informações, consulte Dados em um gráfico de comportamento na documentação do Detective.

sistema big-endian

Um sistema que armazena o byte mais significativo antes. Veja também endianness.

classificação binária

Um processo que prevê um resultado binário (uma de duas classes possíveis). Por exemplo, seu modelo de ML pode precisar prever problemas como “Este e-mail é ou não é spam?” ou “Este produto é um livro ou um carro?”

filtro de bloom

Uma estrutura de dados probabilística e eficiente em termos de memória que é usada para testar se um elemento é membro de um conjunto.

blue/green deployment (implantação azul/verde)

Uma estratégia de implantação em que você cria dois ambientes separados, mas idênticos. Você executa a versão atual da aplicação em um ambiente (azul) e a nova versão da aplicação no outro ambiente (verde). Essa estratégia ajuda você a reverter rapidamente com o mínimo de impacto.

bot

Uma aplicação de software que executa tarefas automatizadas na internet e simula a atividade ou interação humana. Alguns bots são úteis ou benéficos, como crawlers da web que indexam informações na internet. Outros bots, conhecidos como bots maliciosos, têm como objetivo causar disrupção ou danos a indivíduos ou organizações.

botnet

Redes de bots infectadas por malware e sob o controle de uma única parte, conhecidas como bot herder ou operador de bots. Os botnets são o mecanismo mais conhecido para escalar bots e seu impacto.

ramo

Uma área contida de um repositório de código. A primeira ramificação criada em um repositório é a ramificação principal. Você pode criar uma nova ramificação a partir de uma ramificação existente e, em seguida, desenvolver recursos ou corrigir bugs na nova ramificação. Uma ramificação que você cria para gerar um recurso é comumente chamada de ramificação de recurso. Quando o recurso estiver pronto para lançamento, você mesclará a ramificação do recurso de volta com a ramificação principal. Para obter mais informações, consulte Sobre filiais (GitHub documentação).

Acesso de emergência

Em circunstâncias excepcionais e por meio de um processo aprovado, um meio rápido para um usuário obter acesso a um Conta da AWS que ele normalmente não tem permissão para acessar. Para obter mais informações, consulte o indicador Implement break-glass procedures nas orientações do AWS Well-Architected.

estratégia brownfield

A infraestrutura existente em seu ambiente. Ao adotar uma estratégia brownfield para uma arquitetura de sistema, você desenvolve a arquitetura de acordo com as restrições dos sistemas e da infraestrutura atuais. Se estiver expandindo a infraestrutura existente, poderá combinar as estratégias brownfield e greenfield.

cache do buffer

A área da memória em que os dados acessados com mais frequência são armazenados.

capacidade de negócios

O que uma empresa faz para gerar valor (por exemplo, vendas, atendimento ao cliente ou marketing). As arquiteturas de microsserviços e as decisões de desenvolvimento podem ser orientadas por recursos de negócios. Para obter mais informações, consulte a seção Organizados de acordo com as capacidades de negócios do whitepaper Executar microsserviços conteinerizados na AWS.

planejamento de continuidade de negócios (BCP)

Um plano que aborda o impacto potencial de um evento disruptivo, como uma migração em grande escala, nas operações e permite que uma empresa retome as operações rapidamente.

C

CAF

Veja AWS Cloud Adoption Framework.

implantação canário

O lançamento lento e incremental de uma versão para usuários finais. Quando estiver confiante, você implanta a nova versão e substitui a versão atual por completo.

CCoE

Veja Centro de Excelência da Nuvem.

CDC

Veja captura de dados de alteração.

captura de dados de alterações (CDC)

O processo de rastrear alterações em uma fonte de dados, como uma tabela de banco de dados, e registrar metadados sobre a alteração. É possível usar o CDC para várias finalidades, como auditar ou replicar alterações em um sistema de destino para manter a sincronização.

engenharia do caos

Introduzir intencionalmente falhas ou eventos disruptivos para testar a resiliência de um sistema. Você pode usar AWS Fault Injection Service (AWS FIS) para realizar experimentos que estressam suas AWS cargas de trabalho e avaliar sua resposta.

CI/CD

Veja integração e entrega contínuas.

classificação

Um processo de categorização que ajuda a gerar previsões. Os modelos de ML para problemas de classificação predizem um valor discreto. Os valores discretos são sempre diferentes uns dos outros. Por exemplo, um modelo pode precisar avaliar se há ou não um carro em uma imagem.

criptografia no lado do cliente

Criptografia de dados localmente, antes que o alvo os AWS service (Serviço da AWS) receba.

Centro de excelência em nuvem (CCoE)

Uma equipe multidisciplinar que impulsiona os esforços de adoção da nuvem em toda a organização, incluindo o desenvolvimento de práticas recomendas de nuvem, a mobilização de recursos, o estabelecimento de cronogramas de migração e a liderança da organização em transformações em grande escala. Para obter mais informações, consulte as publicações CCo E no blog de estratégia Nuvem AWS corporativa.

computação em nuvem

A tecnologia de nuvem normalmente usada para armazenamento de dados remoto e gerenciamento de dispositivos de IoT. A computação em nuvem é normalmente conectada à tecnologia de computação de borda.

modelo operacional em nuvem

Em uma organização de TI, o modelo operacional usado para criar, amadurecer e otimizar um ou mais ambientes de nuvem. Para obter mais informações, consulte Criar seu modelo operacional de nuvem.

estágios de adoção da nuvem

As quatro fases pelas quais as organizações normalmente passam ao migrar para a Nuvem AWS:

- Projeto: executar alguns projetos relacionados à nuvem para fins de prova de conceito e aprendizado
- Fundação — Fazer investimentos fundamentais para escalar sua adoção da nuvem (por exemplo, criar uma landing zone, definir um CCo E, estabelecer um modelo de operações)
- Migração: migrar aplicações individuais
- Reinvenção: otimizar produtos e serviços e inovar na nuvem

Esses estágios foram definidos por Stephen Orban na postagem do blog The Journey Toward Cloud-First & the Stages of Adoption no blog de estratégia Nuvem AWS empresarial. Para obter informações sobre como eles se relacionam com a estratégia de AWS migração, consulte o guia de preparação para migração.

CMDB

Veja banco de dados de gerenciamento de configuração.

repositório de código

Um local onde o código-fonte e outros ativos, como documentação, amostras e scripts, são armazenados e atualizados por meio de processos de controle de versão. Os repositórios de nuvem comuns incluem o GitHub ou o Bitbucket Cloud. Cada versão do código é chamada de ramificação. Em uma estrutura de microsserviços, cada repositório é dedicado a uma única peça de funcionalidade. Um único pipeline de CI/CD pode usar vários repositórios.

cache frio

Um cache de buffer que está vazio, não está bem preenchido ou contém dados obsoletos ou irrelevantes. Isso afeta a performance porque a instância do banco de dados deve ler da memória principal ou do disco, um processo que é mais lento do que a leitura do cache do buffer.

dados frios

Dados que raramente são acessados e geralmente são históricos. Ao consultar esse tipo de dados, consultas lentas geralmente são aceitáveis. Mover esses dados para níveis ou classes de armazenamento de baixo desempenho e menos caros pode reduzir os custos.

visão computacional (CV)

Um campo de IA que usa machine learning para analisar e extrair informações de formatos visuais, como vídeos e imagens digitais. Por exemplo, a Amazon SageMaker AI fornece algoritmos de processamento de imagem para CV.

desvio de configuração

Em uma workload, uma alteração de configuração em relação ao estado esperado. Isso pode fazer com que a workload se torne incompatível e, normalmente, é gradual e não intencional.

banco de dados de gerenciamento de configuração (CMDB)

Um repositório que armazena e gerencia informações sobre um banco de dados e seu ambiente de TI, incluindo componentes de hardware e software e suas configurações. Normalmente, os dados de um CMDB são usados no estágio de descoberta e análise do portfólio da migração.

pacote de conformidade

Um conjunto de AWS Config regras e ações de remediação que você pode montar para personalizar suas verificações de conformidade e segurança. Você pode implantar um pacote de conformidade como uma entidade única em uma Conta da AWS região ou em uma organização usando um modelo YAML. Para obter mais informações, consulte Pacotes de conformidade na documentação. AWS Config

integração contínua e entrega contínua (CI/CD)

O processo de automatizar os estágios de origem, criação, teste, preparação e produção do processo de lançamento do software. CI/CD é comumente descrito como um pipeline. CI/CD pode ajudá-lo a automatizar processos, melhorar a produtividade, melhorar a qualidade do código e entregar com mais rapidez. Para obter mais informações, consulte Benefícios da entrega contínua. CD também pode significar implantação contínua. Para obter mais informações, consulte Entrega contínua versus implantação contínua.

CV

Veja visão computacional.

D

dados em repouso

Dados estacionários em sua rede, por exemplo, dados que estão em um armazenamento.

classificação de dados

Um processo para identificar e categorizar os dados em sua rede com base em criticalidade e confidencialidade. É um componente crítico de qualquer estratégia de gerenciamento de riscos de

segurança cibernética, pois ajuda a determinar os controles adequados de proteção e retenção para os dados. A classificação de dados é um componente do pilar de segurança no AWS Well-Architected Framework. Para obter mais informações, consulte Classificação de dados.

desvio de dados

Uma variação significativa entre os dados de produção e os dados usados para treinar um modelo de ML ou uma alteração significativa nos dados de entrada ao longo do tempo. O desvio de dados pode reduzir a qualidade geral, a precisão e a imparcialidade das previsões do modelo de ML.

dados em trânsito

Dados que estão se movendo ativamente pela sua rede, como entre os recursos da rede.

data mesh

Um framework de arquitetura que fornece propriedade de dados distribuída e descentralizada com gerenciamento e governança centralizados.

minimização de dados

O princípio de coletar e processar apenas os dados estritamente necessários. Praticar a minimização de dados no Nuvem AWS pode reduzir os riscos de privacidade, os custos e a pegada de carbono de sua análise.

perímetro de dados

Um conjunto de proteções preventivas em seu AWS ambiente que ajudam a garantir que somente identidades confiáveis acessem recursos confiáveis das redes esperadas. Para obter mais informações, consulte Construindo um perímetro de dados em. AWS

pré-processamento de dados

A transformação de dados brutos em um formato que seja facilmente analisado por seu modelo de ML. O pré-processamento de dados pode significar a remoção de determinadas colunas ou linhas e o tratamento de valores ausentes, inconsistentes ou duplicados.

proveniência dos dados

O processo de rastrear a origem e o histórico dos dados ao longo de seu ciclo de vida, por exemplo, como os dados foram gerados, transmitidos e armazenados.

titular dos dados

Um indivíduo cujos dados estão sendo coletados e processados.

data warehouse

Um sistema de gerenciamento de dados compatível com business intelligence, como analytics. Os data warehouses geralmente contêm grandes quantidades de dados históricos e geralmente são usados para consultas e análises.

linguagem de definição de dados (DDL)

Instruções ou comandos para criar ou modificar a estrutura de tabelas e objetos em um banco de dados.

linguagem de manipulação de dados (DML)

Instruções ou comandos para modificar (inserir, atualizar e excluir) informações em um banco de dados.

DDL

Veja linguagem de definição de banco de dados.

deep ensemble

A combinação de vários modelos de aprendizado profundo para gerar previsões. Os deep ensembles podem ser usados para produzir uma previsão mais precisa ou para estimar a incerteza nas previsões.

Aprendizado profundo

Um subcampo do ML que usa várias camadas de redes neurais artificiais para identificar o mapeamento entre os dados de entrada e as variáveis-alvo de interesse.

defense-in-depth

Uma abordagem de segurança da informação na qual uma série de mecanismos e controles de segurança são cuidadosamente distribuídos por toda a rede de computadores para proteger a confidencialidade, a integridade e a disponibilidade da rede e dos dados nela contidos. Ao adotar essa estratégia AWS, você adiciona vários controles em diferentes camadas da AWS Organizations estrutura para ajudar a proteger os recursos. Por exemplo, uma defense-in-depth abordagem pode combinar autenticação multifatorial, segmentação de rede e criptografia.

administrador delegado

Em AWS Organizations, um serviço compatível pode registrar uma conta de AWS membro para administrar as contas da organização e gerenciar as permissões desse serviço. Essa conta

é chamada de administrador delegado para esse serviço Para obter mais informações e uma lista de serviços compatíveis, consulte Serviços que funcionam com o AWS Organizations na documentação do AWS Organizations .

implantação

O processo de criar uma aplicação, novos recursos ou correções de código disponíveis no ambiente de destino. A implantação envolve a implementação de mudanças em uma base de código e, em seguida, a criação e execução dessa base de código nos ambientes da aplicação

ambiente de desenvolvimento

Veja ambiente.

controle detectivo

Um controle de segurança projetado para detectar, registrar e alertar após a ocorrência de um evento. Esses controles são uma segunda linha de defesa, alertando você sobre eventos de segurança que contornaram os controles preventivos em vigor. Para obter mais informações, consulte Controles detectivos em Como implementar controles de segurança na AWS.

mapeamento do fluxo de valor de desenvolvimento (DVSM)

Um processo usado para identificar e priorizar restrições que afetam negativamente a velocidade e a qualidade em um ciclo de vida de desenvolvimento de software. O DVSM estende o processo de mapeamento do fluxo de valor originalmente projetado para práticas de manufatura enxuta. Ele se concentra nas etapas e equipes necessárias para criar e movimentar valor por meio do processo de desenvolvimento de software.

gêmeo digital

Uma representação virtual de um sistema real, como um prédio, fábrica, equipamento industrial ou linha de produção. Os gêmeos digitais oferecem suporte à manutenção preditiva, ao monitoramento remoto e à otimização da produção.

tabela de dimensões

Em um esquema em estrela, uma tabela menor que contém atributos de dados sobre dados quantitativos em uma tabela de fatos. Os atributos da tabela de dimensões geralmente são campos de texto ou números discretos que se comportam como texto. Esses atributos normalmente são usados para restringir consultas, filtrar e rotular conjuntos de resultados.

desastre

Um evento que impede que uma workload ou sistema cumpra seus objetivos de negócios em seu local principal de implantação. Esses eventos podem ser desastres naturais, falhas técnicas ou o resultado de ações humanas, como configuração incorreta não intencional ou ataque de malware.

Recuperação de desastres (RD)

A estratégia e o processo que você usa para minimizar o tempo de inatividade e a perda de dados causados por um disastre. Para obter mais informações, consulte Recuperação de desastres de cargas de trabalho em AWS: Recuperação na nuvem no AWS Well-Architected Framework.

DML

Veja linguagem de manipulação de banco de dados.

design orientado por domínio

Uma abordagem ao desenvolvimento de um sistema de software complexo conectando seus componentes aos domínios em evolução, ou principais metas de negócios, atendidos por cada componente. Esse conceito foi introduzido por Eric Evans em seu livro, Design orientado por domínio: lidando com a complexidade no coração do software (Boston: Addison-Wesley Professional, 2003). Para obter informações sobre como usar o design orientado por domínio com o padrão strangler fig, consulte Modernizar incrementalmente os serviços web herdados do Microsoft ASP.NET (ASMX) usando contêineres e o Amazon API Gateway.

DR

Veja recuperação de desastres.

Detecção da oscilação

Rastreamento de desvios de uma configuração de linha de base. Por exemplo, você pode usar AWS CloudFormation para detectar desvios nos recursos do sistema ou AWS Control Tower para detectar mudanças em seu landing zone que possam afetar a conformidade com os requisitos de governança.

DVSM

Veja mapeamento do fluxo de valor de desenvolvimento.

E

EDA

Veja análise exploratória de dados.

EDI

Veja intercâmbio eletrônico de dados.

computação de borda

A tecnologia que aumenta o poder computacional de dispositivos inteligentes nas bordas de uma rede de IoT. Quando comparada com a computação em nuvem, a computação de borda pode reduzir a latência da comunicação e melhorar o tempo de resposta.

intercâmbio eletrônico de dados (EDI)

A troca automatizada de documentos comerciais entre organizações. Para obter mais informações, consulte O que é EDI (Intercâmbio eletrônico de dados)?.

criptografia

Um processo de computação que transforma dados de texto simples, legíveis por humanos, em texto cifrado.

chave de criptografia

Uma sequência criptográfica de bits aleatórios que é gerada por um algoritmo de criptografia. As chaves podem variar em tamanho, e cada chave foi projetada para ser imprevisível e exclusiva.

endianismo

A ordem na qual os bytes são armazenados na memória do computador. Os sistemas big-endian armazenam o byte mais significativo antes. Os sistemas little-endian armazenam o byte menos significativo antes.

endpoint

Veja endpoint de serviço.

serviço de endpoint

Um serviço que pode ser hospedado em uma nuvem privada virtual (VPC) para ser compartilhado com outros usuários. Você pode criar um serviço de endpoint com AWS PrivateLink e conceder permissões a outros diretores Contas da AWS ou a AWS Identity and Access Management (IAM).

Essas contas ou entidades principais podem se conectar ao serviço de endpoint de maneira privada criando endpoints da VPC de interface. Para obter mais informações, consulte Criar um serviço de endpoint na documentação do Amazon Virtual Private Cloud (Amazon VPC).

planejamento de recursos empresariais (ERP)

Um sistema que automatiza e gerencia os principais processos de negócios (como contabilidade, MES e gerenciamento de projetos) para uma empresa.

criptografia envelopada

O processo de criptografar uma chave de criptografia com outra chave de criptografia. Para obter mais informações, consulte Criptografia de envelope na documentação AWS Key Management Service (AWS KMS).

ambiente

Uma instância de uma aplicação em execução. Estes são tipos comuns de ambientes na computação em nuvem:

- ambiente de desenvolvimento: uma instância de uma aplicação em execução que está disponível somente para a equipe principal responsável pela manutenção da aplicação. Ambientes de desenvolvimento são usados para testar mudanças antes de promovê-las para ambientes superiores. Esse tipo de ambiente às vezes é chamado de ambiente de teste.
- ambientes inferiores: todos os ambientes de desenvolvimento para uma aplicação, como aqueles usados para compilações e testes iniciais.
- ambiente de produção: uma instância de uma aplicação em execução que os usuários finais podem acessar. Em um CI/CD pipeline, o ambiente de produção é o último ambiente de implantação.
- ambientes superiores: todos os ambientes que podem ser acessados por usuários que não sejam a equipe principal de desenvolvimento. Isso pode incluir um ambiente de produção, ambientes de pré-produção e ambientes para testes de aceitação do usuário.

epic

Em metodologias ágeis, categorias funcionais que ajudam a organizar e priorizar seu trabalho. Os epics fornecem uma descrição de alto nível dos requisitos e das tarefas de implementação. Por exemplo, os épicos de segurança AWS da CAF incluem gerenciamento de identidade e acesso, controles de detetive, segurança de infraestrutura, proteção de dados e resposta a incidentes. Para obter mais informações sobre epics na estratégia de migração da AWS , consulte o guia de implementação do programa.

ERP

Veja planejamento de recursos empresariais.

análise exploratória de dados (EDA)

O processo de analisar um conjunto de dados para entender suas principais características. Você coleta ou agrega dados e, em seguida, realiza investigações iniciais para encontrar padrões, detectar anomalias e verificar suposições. O EDA é realizado por meio do cálculo de estatísticas resumidas e da criação de visualizações de dados.

F

tabela de fatos

A tabela central em um esquema em estrela. Ela armazena dados quantitativos sobre as operações comerciais. Normalmente, uma tabela de fatos contém dois tipos de colunas: as que contêm medidas e as que contêm uma chave externa para uma tabela de dimensões.

Antecipar-se à falha

Uma filosofia que usa testes frequentes e incrementais para reduzir o ciclo de vida do desenvolvimento. É uma parte essencial de uma abordagem ágil.

delimitação de isolamento contra falhas

No Nuvem AWS, um limite, como uma zona de disponibilidade, Região da AWS um plano de controle ou um plano de dados, que limita o efeito de uma falha e ajuda a melhorar a resiliência das cargas de trabalho. Para obter mais informações, consulte AWS Fault Isolation Boundaries.

ramificação de recursos

Veja ramificação.

recursos

Os dados de entrada usados para fazer uma previsão. Por exemplo, em um contexto de manufatura, os recursos podem ser imagens capturadas periodicamente na linha de fabricação.

importância do recurso

O quanto um recurso é importante para as previsões de um modelo. Isso geralmente é expresso como uma pontuação numérica que pode ser calculada por meio de várias técnicas, como

Shapley Additive Explanations (SHAP) e gradientes integrados. Para obter mais informações, consulte Interpretabilidade do modelo de aprendizado de máquina com AWS.

transformação de recursos

O processo de otimizar dados para o processo de ML, incluindo enriquecer dados com fontes adicionais, escalar valores ou extrair vários conjuntos de informações de um único campo de dados. Isso permite que o modelo de ML se beneficie dos dados. Por exemplo, se a data “2021-05-27 00:15:37” for dividida em “2021”, “maio”, “quinta” e “15”, isso poderá ajudar o algoritmo de aprendizado a aprender padrões diferenciados associados a diferentes componentes de dados.

prompt few shot

Fornecer a um LLM um pequeno número de exemplos que demonstram a tarefa e o resultado desejado antes de solicitar que ele execute uma tarefa semelhante. Essa técnica é uma aplicação do aprendizado em contexto, em que os modelos aprendem com exemplos (shots) incorporados aos prompts. Prompts few-shot podem ser eficazes para tarefas que exigem formatação, raciocínio ou conhecimento de domínio específicos. Veja também prompts zero-shot.

FGAC

Veja controle de acesso refinado.

Controle de acesso refinado (FGAC)

O uso de várias condições para permitir ou negar uma solicitação de acesso.

migração flash-cut

Um método de migração de banco de dados que usa replicação contínua de dados via captura de dados de alteração para migrar os dados no menor tempo possível, em vez de usar uma abordagem em fases. O objetivo é reduzir ao mínimo o tempo de inatividade.

FM

Veja modelo de base.

modelo de base (FM)

Uma grande rede neural de aprendizado profundo que vem treinando em grandes conjuntos de dados generalizados e não rotulados. FMs são capazes de realizar uma ampla variedade de tarefas gerais, como entender a linguagem, gerar texto e imagens e conversar em linguagem natural. Para obter mais informações, consulte O que são modelos de base?.

G

IA generativa

Um subconjunto de modelos de IA que foram treinados em grandes quantidades de dados e que podem usar um simples prompt de texto para criar novos artefatos e conteúdo, como imagens, vídeos, texto e áudio. Para obter mais informações, consulte O que é IA generativa?.

bloqueio geográfico

Veja restrições geográficas.

restrições geográficas (bloqueio geográfico)

Na Amazon CloudFront, uma opção para impedir que usuários em países específicos acessem distribuições de conteúdo. É possível usar uma lista de permissões ou uma lista de bloqueios para especificar países aprovados e banidos. Para obter mais informações, consulte Restringir a distribuição geográfica do seu conteúdo na CloudFront documentação.

Fluxo de trabalho do GitFlow

Uma abordagem na qual ambientes inferiores e superiores usam ramificações diferentes em um repositório de código-fonte. O fluxo de trabalho do Gitflow é considerado legado, e o fluxo de trabalho trunk-based é a abordagem moderna e preferencial.

golden image

Um snapshot de um sistema ou software usado como modelo para implantar novas instâncias desse sistema ou software. Por exemplo, na manufatura, uma golden image pode ser usada para provisionar software em vários dispositivos e ajudar a melhorar a velocidade, a escalabilidade e a produtividade nas operações de fabricação de dispositivos.

estratégia greenfield

A ausência de infraestrutura existente em um novo ambiente. Ao adotar uma estratégia greenfield para uma arquitetura de sistema, é possível selecionar todas as novas tecnologias sem a restrição da compatibilidade com a infraestrutura existente, também conhecida como brownfield. Se estiver expandindo a infraestrutura existente, poderá combinar as estratégias brownfield e greenfield.

barreira de proteção

Uma regra de alto nível que ajuda a governar recursos, políticas e conformidade em todas as unidades organizacionais ()OUs. Barreiras de proteção preventivas impõem políticas para

garantir o alinhamento a padrões de conformidade. Elas são implementadas usando políticas de controle de serviço e limites de permissões do IAM. Barreiras de proteção detectivas detectam violações de políticas e problemas de conformidade e geram alertas para remediação. Eles são implementados usando AWS Config, AWS Security Hub CSPM, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector e verificações personalizadas AWS Lambda .

H

HA

Veja alta disponibilidade.

migração heterogênea de bancos de dados

Migrar seu banco de dados de origem para um banco de dados de destino que usa um mecanismo de banco de dados diferente (por exemplo, Oracle para Amazon Aurora). A migração heterogênea geralmente faz parte de um esforço de redefinição da arquitetura, e converter o esquema pode ser uma tarefa complexa. O AWS fornece o AWS SCT para ajudar nas conversões de esquemas.

alta disponibilidade (HA)

A capacidade de uma workload operar continuamente, sem intervenção, em caso de desafios ou desastres. Os sistemas AH são projetados para realizar o failover automático, oferecer consistentemente desempenho de alta qualidade e lidar com diferentes cargas e falhas com impacto mínimo no desempenho.

modernização de historiador

Uma abordagem usada para modernizar e atualizar os sistemas de tecnologia operacional (OT) para melhor atender às necessidades do setor de manufatura. Um historiador é um tipo de banco de dados usado para coletar e armazenar dados de várias fontes em uma fábrica.

dados de hold-out

Uma parte dos dados históricos rotulados que são retidos de um conjunto de dados usado para treinar um modelo de machine learning. Você pode usar dados de hold-out para avaliar a performance do modelo comparando as predições do modelo com os dados de retenção.

migração homogênea de bancos de dados

Migrar seu banco de dados de origem para um banco de dados de destino que compartilha o mesmo mecanismo de banco de dados (por exemplo, Microsoft SQL Server para Amazon RDS para SQL Server). A migração homogênea geralmente faz parte de um esforço de redefinição da hospedagem ou da plataforma. É possível usar utilitários de banco de dados nativos para migrar o esquema.

dados quentes

Dados acessados com frequência, como dados em tempo real ou dados translacionais recentes. Esses dados normalmente exigem uma camada ou classe de armazenamento de alto desempenho para fornecer respostas rápidas às consultas.

hotfix

Uma correção urgente para um problema crítico em um ambiente de produção. Devido à sua urgência, um hotfix geralmente é feito fora do fluxo de trabalho normal de DevOps lançamento.

período de hipercuidados

Imediatamente após a substituição, o período em que uma equipe de migração gerencia e monitora as aplicações migradas na nuvem para resolver quaisquer problemas. Normalmente, a duração desse período é de 1 a 4 dias. No final do período de hipercuidados, a equipe de migração normalmente transfere a responsabilidade pelas aplicações para a equipe de operações de nuvem.

eu

IaC

Veja infraestrutura como código.

Política baseada em identidade

Uma política anexada a um ou mais diretores do IAM que define suas permissões no Nuvem AWS ambiente.

aplicação ociosa

Uma aplicação que tem um uso médio de CPU e memória entre 5 e 20% em um período de 90 dias. Em um projeto de migração, é comum retirar essas aplicações ou retê-las on-premises.

IIoT

Veja Internet das Coisas Industrial.

infraestrutura imutável

Um modelo que implanta uma nova infraestrutura para workloads de produção em vez de atualizar, aplicar patches ou modificar a infraestrutura existente. Infraestruturas imutáveis são inerentemente mais consistentes, confiáveis e preditivas do que infraestruturas mutáveis. Para obter mais informações, consulte a prática recomendada Implantar usando infraestrutura imutável no AWS Well-Architected Framework.

VPC de entrada (admissão)

Em uma arquitetura de AWS várias contas, uma VPC que aceita, inspeciona e roteia conexões de rede de fora de um aplicativo. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

migração incremental

Uma estratégia de substituição na qual você migra a aplicação em pequenas partes, em vez de realizar uma única substituição completa. Por exemplo, é possível mover inicialmente apenas alguns microsserviços ou usuários para o novo sistema. Depois de verificar se tudo está funcionando corretamente, mova os microsserviços ou usuários adicionais de forma incremental até poder descomissionar seu sistema herdado. Essa estratégia reduz os riscos associados a migrações de grande porte.

Indústria 4.0

Um termo que foi introduzido por Klaus Schwab em 2016 para se referir à modernização dos processos de manufatura por meio de avanços em conectividade, dados em tempo real, automação, analytics e IA/ML.

infraestrutura

Todos os recursos e ativos contidos no ambiente de uma aplicação.

Infraestrutura como código (IaC)

O processo de provisionamento e gerenciamento da infraestrutura de uma aplicação por meio de um conjunto de arquivos de configuração. A IaC foi projetada para ajudar você a centralizar o gerenciamento da infraestrutura, padronizar recursos e escalar rapidamente para que novos ambientes sejam reproduzíveis, confiáveis e consistentes.

Internet industrial das coisas (IIoT)

O uso de sensores e dispositivos conectados à Internet nos setores industriais, como manufatura, energia, automotivo, saúde, ciências biológicas e agricultura. Para obter mais informações, consulte Criando uma estratégia de transformação digital industrial da Internet das Coisas (IIoT).

VPC de inspeção

Em uma arquitetura de AWS várias contas, uma VPC centralizada que gerencia as inspeções do tráfego de rede entre VPCs (na mesma ou em diferentes Regiões da AWS) a Internet e as redes locais. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

Internet das coisas (IoT)

A rede de objetos físicos conectados com sensores ou processadores incorporados que se comunicam com outros dispositivos e sistemas pela Internet ou por uma rede de comunicação local. Para obter mais informações, consulte O que é IoT?

interpretabilidade

Uma característica de um modelo de machine learning que descreve o grau em que um ser humano pode entender como as previsões do modelo dependem de suas entradas. Para obter mais informações, consulte Interpretabilidade do modelo de aprendizado de máquina com AWS.

IoT

Veja Internet das Coisas.

Biblioteca de informações de TI (ITIL)

Um conjunto de práticas recomendadas para fornecer serviços de TI e alinhar esses serviços a requisitos de negócios. A ITIL fornece a base para o ITSM.

Gerenciamento de serviços de TI (ITSM)

Atividades associadas a design, implementação, gerenciamento e suporte de serviços de TI para uma organização. Para obter informações sobre a integração de operações em nuvem com ferramentas de ITSM, consulte o guia de integração de operações.

ITIL

Veja biblioteca de informações de TI.

ITSM

Veja gerenciamento de serviços de TI.

L

controle de acesso baseado em etiqueta (LBAC)

Uma implementação do controle de acesso obrigatório (MAC) em que os usuários e os dados em si recebem explicitamente um valor de etiqueta de segurança. A interseção entre a etiqueta de segurança do usuário e a etiqueta de segurança dos dados determina quais linhas e colunas podem ser vistas pelo usuário.

zona de pouso

Uma landing zone é um AWS ambiente bem arquitetado, com várias contas, escalável e seguro. Um ponto a partir do qual suas organizações podem iniciar e implantar rapidamente workloads e aplicações com confiança em seu ambiente de segurança e infraestrutura. Para obter mais informações sobre zonas de pouso, consulte Configurar um ambiente da AWS com várias contas seguro e escalável.

grande modelo de linguagem (LLM)

Um modelo de IA de aprendizado profundo pré-treinado em uma grande quantidade de dados. Um LLM pode realizar várias tarefas, como responder a perguntas, resumir documentos, traduzir texto para outros idiomas e completar frases. Para obter mais informações, consulte O que são LLMs.

migração de grande porte

Uma migração de 300 servidores ou mais.

LBAC

Veja controle de acesso baseado em rótulo.

privilégio mínimo

A prática recomendada de segurança de conceder as permissões mínimas necessárias para executar uma tarefa. Para obter mais informações, consulte Aplicar permissões de privilégios mínimos na documentação do IAM.

mover sem alterações (lift-and-shift)

Veja 7 Rs.

sistema little-endian

Um sistema que armazena o byte menos significativo antes. Veja também endianness.

LLM

Veja grande modelo de linguagem.

ambientes inferiores

Veja ambiente.

M

machine learning (ML)

Um tipo de inteligência artificial que usa algoritmos e técnicas para reconhecimento e aprendizado de padrões. O ML analisa e aprende com dados gravados, por exemplo, dados da Internet das Coisas (IoT), para gerar um modelo estatístico baseado em padrões. Para obter mais informações, consulte Machine learning.

ramificação principal

Veja ramificação.

Malware

Software projetado para comprometer a segurança ou a privacidade do computador. O malware pode interromper os sistemas do computador, vazar informações sensíveis ou obter acesso não autorizado. Exemplos de malware incluem vírus, worms, ransomware, cavalos de Troia, spyware e keyloggers.

Serviços gerenciados

Serviços da AWS para o qual AWS opera a camada de infraestrutura, o sistema operacional e as plataformas, e você acessa os endpoints para armazenar e recuperar dados. O Amazon Simple Storage Service (Amazon S3) e o Amazon DynamoDB são exemplos de serviços gerenciados. Eles também são conhecidos como serviços abstraídos.

sistema de execução de manufatura (MES)

Um sistema de software para rastrear, monitorar, documentar e controlar processos de produção que convertem matérias-primas em produtos acabados no chão de fábrica.

MAP

Veja Programa de Aceleração da Migração.

mecanismo

Um processo completo em que você cria uma ferramenta, impulsiona a adoção da ferramenta e, em seguida, inspeciona os resultados para fazer ajustes. Um mecanismo é um ciclo que se reforça e se aprimora à medida que opera. Para obter mais informações, consulte Construindo mecanismos no AWS Well-Architected Framework.

conta de membro

Todos, Contas da AWS exceto a conta de gerenciamento, que fazem parte de uma organização em AWS Organizations. Uma conta só pode ser membro de uma organização de cada vez.

MES

Veja sistema de execução de manufatura.

Transporte de Telemetria de Enfileiramento de Mensagens (MQTT)

Um protocolo de comunicação leve machine-to-machine (M2M), baseado no padrão de publicação/assinatura, para dispositivos de IoT com recursos limitados.

microsserviço

Um serviço pequeno e independente que se comunica de forma bem definida APIs e normalmente é de propriedade de equipes pequenas e independentes. Por exemplo, um sistema de seguradora pode incluir microsserviços que mapeiam as capacidades comerciais, como vendas ou marketing, ou subdomínios, como compras, reclamações ou análises. Os benefícios dos microsserviços incluem agilidade, escalabilidade flexível, fácil implantação, código reutilizável e resiliência. Para obter mais informações, consulte Integração de microsserviços usando serviços sem AWS servidor.

arquitetura de microsserviços

Uma abordagem à criação de aplicações com componentes independentes que executam cada processo de aplicação como um microsserviço. Esses microsserviços se comunicam por meio

de uma interface bem definida usando leveza. APIs Cada microsserviço nessa arquitetura pode ser atualizado, implantado e escalado para atender à demanda por funções específicas de uma aplicação. Para obter mais informações, consulte Implementação de microsserviços em. AWS

Programa de Aceleração da Migração (MAP)

Um AWS programa que fornece suporte de consultoria, treinamento e serviços para ajudar as organizações a criar uma base operacional sólida para migrar para a nuvem e ajudar a compensar o custo inicial das migrações. O MAP inclui uma metodologia de migração para executar migrações legadas de forma metódica e um conjunto de ferramentas para automatizar e acelerar cenários comuns de migração.

migração em escala

O processo de mover a maior parte do portfólio de aplicações para a nuvem em ondas, com mais aplicações sendo movidas em um ritmo mais rápido a cada onda. Essa fase usa as práticas recomendadas e lições aprendidas nas fases anteriores para implementar uma fábrica de migração de equipes, ferramentas e processos para agilizar a migração de workloads por meio de automação e entrega ágeis. Esta é a terceira fase da estratégia de migração para a AWS.

fábrica de migração

Equipes multifuncionais que simplificam a migração de workloads por meio de abordagens automatizadas e ágeis. As equipes da fábrica de migração geralmente incluem operações, analistas e proprietários de negócios, engenheiros de migração, desenvolvedores e DevOps profissionais que trabalham em sprints. Entre 20 e 50% de um portfólio de aplicações corporativas consiste em padrões repetidos que podem ser otimizados por meio de uma abordagem de fábrica. Para obter mais informações, consulte discussão sobre fábricas de migração e o guia do Cloud Migration Factory neste conjunto de conteúdo.

metadados de migração

As informações sobre a aplicação e o servidor necessárias para concluir a migração. Cada padrão de migração exige um conjunto de metadados de migração diferente. Exemplos de metadados de migração incluem a sub-rede, o grupo de segurança e AWS a conta de destino.

padrão de migração

Uma tarefa de migração repetível que detalha a estratégia de migração, o destino da migração e a aplicação ou o serviço de migração usado. Exemplo: rehospede a migração para o Amazon EC2 AWS com o Application Migration Service.

Avaliação de Portfólio para Migração (MPA)

Uma ferramenta on-line que fornece informações para validar o caso de negócios para migrar para a Nuvem AWS. O MPA fornece avaliação detalhada do portfólio (dimensionamento correto do servidor, preços, comparações de TCO, análise de custos de migração), bem como planejamento de migração (análise e coleta de dados de aplicações, agrupamento de aplicações, priorização de migração e planejamento de ondas). A ferramenta MPA (requer login) está disponível gratuitamente para todos os AWS consultores e consultores parceiros da APN.

Avaliação de Preparação para Migração (MRA)

O processo de obter insights sobre o status de prontidão de uma organização para a nuvem, identificar pontos fortes e fracos e criar um plano de ação para fechar as lacunas identificadas, usando o CAF. AWS Para mais informações, consulte o guia de preparação para migração. A MRA é a primeira fase da estratégia de migração para a AWS.

estratégia de migração

A abordagem usada para migrar uma workload para a Nuvem AWS. Para obter mais informações, veja a entrada 7 Rs neste glossário e consulte Mobilize sua organização para acelerar migrações em grande escala.

ML

Veja machine learning.

modernização

Transformar uma aplicação desatualizada (herdada ou monolítica) e sua infraestrutura em um sistema ágil, elástico e altamente disponível na nuvem para reduzir custos, ganhar eficiência e aproveitar as inovações. Para obter mais informações, consulte Strategy for modernizing applications in the Nuvem AWS.

avaliação de preparação para modernização

Uma avaliação que ajuda a determinar a preparação para modernização das aplicações de uma organização. Ela identifica benefícios, riscos e dependências e determina o quão bem a organização pode acomodar o estado futuro dessas aplicações. O resultado da avaliação é um esquema da arquitetura de destino, um roteiro que detalha as fases de desenvolvimento e os marcos do processo de modernização e um plano de ação para abordar as lacunas identificadas. Para obter mais informações, consulte Evaluating modernization readiness for applications in the Nuvem AWS.

aplicações monolíticas (monólitos)

Aplicações que são executadas como um único serviço com processos fortemente acoplados. As aplicações monolíticas apresentam várias desvantagens. Se um recurso da aplicação apresentar um aumento na demanda, toda a arquitetura deverá ser escalada. Adicionar ou melhorar os recursos de uma aplicação monolítica também se torna mais complexo quando a base de código cresce. Para resolver esses problemas, é possível criar uma arquitetura de microsserviços. Para obter mais informações, consulte Decompor monólitos em microsserviços.

MPA

Veja Avaliação do Portfólio para Migração.

MQTT

Veja Transporte de Telemetria de Enfileiramento de Mensagens.

classificação multiclasse

Um processo que ajuda a gerar previsões para várias classes (prevendo um ou mais de dois resultados). Por exemplo, um modelo de ML pode perguntar “Este produto é um livro, um carro ou um telefone?” ou “Qual categoria de produtos é mais interessante para este cliente?”

infraestrutura mutável

Um modelo que atualiza e modifica a infraestrutura existente para workloads de produção. Para melhorar a consistência, confiabilidade e previsibilidade, o AWS Well-Architected Framework recomenda o uso de infraestrutura imutável como uma prática recomendada.

O

OAC

Veja controle de acesso de origem.

OAI

Veja identidade de acesso de origem.

OCM

Veja gerenciamento de alterações organizacionais.

migração offline

Um método de migração no qual a workload de origem é desativada durante o processo de migração. Esse método envolve tempo de inatividade prolongado e geralmente é usado para workloads pequenas e não críticas.

OI

Veja integração de operações.

Ola

Veja acordo de nível operacional.

migração online

Um método de migração no qual a workload de origem é copiada para o sistema de destino sem ser colocada offline. As aplicações conectadas à workload podem continuar funcionando durante a migração. Esse método envolve um tempo de inatividade nulo ou mínimo e normalmente é usado para workloads essenciais para a produção.

OPC-UA

Veja Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

Um protocolo de comunicação machine-to-machine (M2M) para automação industrial. O OPC-UA fornece um padrão de interoperabilidade com esquemas de criptografia, autenticação e autorização de dados.

acordo de nível operacional (OLA)

Um acordo que esclarece o que os grupos funcionais de TI prometem oferecer uns aos outros para apoiar um acordo de serviço (SLA).

análise de prontidão operacional (ORR)

Uma lista de verificação de perguntas e práticas recomendadas associadas que ajudam você a entender, avaliar, prevenir ou reduzir o escopo de incidentes e possíveis falhas. Para obter mais informações, consulte Operational Readiness Reviews (ORR) no AWS Well-Architected Framework.

tecnologia operacional (TO)

Sistemas de hardware e software que trabalham com o ambiente físico para controlar operações, equipamentos e infraestrutura industriais. Na manufatura, a integração dos sistemas de

tecnologia da informação (TI) e tecnologia operacional (TO) é o foco principal das transformações da Indústria 4.0.

integração de operações (OI)

O processo de modernização das operações na nuvem, que envolve planejamento de preparação, automação e integração. Para obter mais informações, consulte o guia de integração de operações.

trilha organizacional

Uma trilha criada por ela AWS CloudTrail registra todos os eventos de todos Contas da AWS em uma organização em AWS Organizations. Essa trilha é criada em cada Conta da AWS que faz parte da organização e monitora a atividade em cada conta. Para obter mais informações, consulte Criação de uma trilha para uma organização na CloudTrail documentação.

gerenciamento de alterações organizacionais (OCM)

Uma estrutura para gerenciar grandes transformações de negócios disruptivas de uma perspectiva de pessoas, cultura e liderança. O OCM ajuda as organizações a se prepararem e fazerem a transição para novos sistemas e estratégias, acelerando a adoção de alterações, abordando questões de transição e promovendo mudanças culturais e organizacionais. Na estratégia de AWS migração, essa estrutura é chamada de aceleração de pessoas, devido à velocidade de mudança exigida nos projetos de adoção da nuvem. Para obter mais informações, consulte o guia do OCM.

controle de acesso de origem (OAC)

Em CloudFront, uma opção aprimorada para restringir o acesso para proteger seu conteúdo do Amazon Simple Storage Service (Amazon S3). O OAC oferece suporte a todos os buckets S3 Regiões da AWS, criptografia do lado do servidor com AWS KMS (SSE-KMS) e solicitações dinâmicas ao bucket S3. `PUT` `DELETE`

Identidade do acesso de origem (OAI)

Em CloudFront, uma opção para restringir o acesso para proteger seu conteúdo do Amazon S3. Quando você usa o OAI, CloudFront cria um principal com o qual o Amazon S3 pode se autenticar. Os diretores autenticados podem acessar o conteúdo em um bucket do S3 somente por meio de uma distribuição específica. CloudFront Veja também OAC, que fornece um controle de acesso mais granular e aprimorado.

ORR

Veja análise de prontidão operacional.

OT

Veja tecnologia operacional.

VPC de saída (egresso)

Em uma arquitetura de AWS várias contas, uma VPC que gerencia conexões de rede que são iniciadas de dentro de um aplicativo. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

P

limite de permissões

Uma política de gerenciamento do IAM anexada a entidades principais do IAM para definir as permissões máximas que o usuário ou perfil podem ter. Para obter mais informações, consulte Limites de permissões na documentação do IAM.

Informações de identificação pessoal (PII)

Informações que, quando visualizadas diretamente ou combinadas com outros dados relacionados, podem ser usadas para inferir razoavelmente a identidade de um indivíduo. Exemplos de PII incluem nomes, endereços e informações de contato.

PII

Veja informações de identificação pessoal.

manual

Um conjunto de etapas predefinidas que capturam o trabalho associado às migrações, como a entrega das principais funções operacionais na nuvem. Um manual pode assumir a forma de scripts, runbooks automatizados ou um resumo dos processos ou etapas necessários para operar seu ambiente modernizado.

PLC

Veja controlador lógico programável.

PLM

Veja gerenciamento do ciclo de vida do produto.

política

Um objeto que pode definir permissões (veja política baseada em identidade), especificar condições de acesso (veja polítca baseada em recurso) ou definir as permissões máximas para todas as contas em uma organização no AWS Organizations (veja política de controle de serviços).

persistência poliglota

Escolher de forma independente a tecnologia de armazenamento de dados de um microsserviço com base em padrões de acesso a dados e outros requisitos. Se seus microsserviços tiverem a mesma tecnologia de armazenamento de dados, eles poderão enfrentar desafios de implementação ou apresentar baixa performance. Os microsserviços serão implementados com mais facilidade e alcançarão performance e escalabilidade melhores se usarem o armazenamento de dados mais bem adaptado às suas necessidades.

avaliação do portfólio

Um processo de descobrir, analisar e priorizar o portfólio de aplicações para planejar a migração. Para obter mais informações, consulte Avaliar a preparação para a migração.

predicado

Uma condição de consulta que retorna `true` ou `false`, normalmente localizada em uma cláusula `WHERE`.

pushdown de predicados

Uma técnica de otimização de consultas de banco de dados que filtra os dados na consulta antes da transferência. Isso reduz a quantidade de dados que devem ser recuperados e processados do banco de dados relacional e melhora a performance das consultas.

controle preventivo

Um controle de segurança projetado para evitar que um evento ocorra. Esses controles são a primeira linha de defesa para ajudar a evitar acesso não autorizado ou alterações indesejadas em sua rede. Para obter mais informações, consulte Controles preventivos em Como implementar controles de segurança na AWS.

principal (entidade principal)

Uma entidade AWS que pode realizar ações e acessar recursos. Essa entidade geralmente é um usuário raiz para um Conta da AWS, uma função do IAM ou um usuário. Para obter mais

informações, consulte Entidade principal em Termos e conceitos de perfis na documentação do IAM.

Privacidade por design

Uma abordagem em engenharia de sistemas que leva em consideração a privacidade em todo o processo de desenvolvimento.

zonas hospedadas privadas

Um contêiner que contém informações sobre como você deseja que o Amazon Route 53 responda às consultas de DNS para um domínio e seus subdomínios em um ou mais. VPCs Para obter mais informações, consulte Como trabalhar com zonas hospedadas privadas na documentação do Route 53.

controle proativo

Um controle de segurança desenvolvido para evitar a implantação de recursos não conformes. Esses controles verificam os recursos antes de serem provisionados. Se o recurso não estiver em conformidade com o controle, ele não será provisionado. Para obter mais informações, consulte o guia de referência de controles na AWS Control Tower documentação e consulte Controles proativos em Implementação de controles de segurança em AWS.

gerenciamento do ciclo de vida do produto (PLM)

O gerenciamento de dados e processos de um produto em todo o seu ciclo de vida, desde a concepção, o desenvolvimento e o lançamento, passando pelo crescimento e maturidade, até o declínio e a remoção.

ambiente de produção

Veja ambiente.

controlador lógico programável (PLC)

Na manufatura, um computador altamente confiável e adaptável que monitora as máquinas e automatiza os processos de fabricação.

encadeamento de prompts

Uso da saída de um prompt do LLM como entrada para o próximo prompt para gerar respostas melhores. Essa técnica é usada para dividir uma tarefa complexa em subtarefas, ou para refinar ou expandir iterativamente uma resposta preliminar. Isso ajuda a melhorar a precisão e a relevância das respostas de um modelo e permite resultados mais granulares e personalizados.

pseudonimização

O processo de substituir identificadores pessoais em um conjunto de dados por valores de espaço reservado. A pseudonimização pode ajudar a proteger a privacidade pessoal. Os dados pseudonimizados ainda são considerados dados pessoais.

publish/subscribe (pub/sub)

Um padrão que permite comunicações assíncronas entre microsserviços para melhorar a escalabilidade e a capacidade de resposta. Por exemplo, em um MES baseado em microsserviços, um microsserviço pode publicar mensagens de eventos em um canal em que outros microsserviços possam assinar. O sistema pode adicionar novos microsserviços sem alterar o serviço de publicação.

Q

plano de consulta

Uma série de etapas, como instruções, usadas para acessar os dados em um sistema de banco de dados relacional SQL.

regressão de planos de consultas

Quando um otimizador de serviço de banco de dados escolhe um plano menos adequado do que escolhia antes de uma determinada alteração no ambiente de banco de dados ocorrer. Isso pode ser causado por alterações em estatísticas, restrições, configurações do ambiente, associações de parâmetros de consulta e atualizações do mecanismo de banco de dados.

R

Matriz RACI

Veja responsável, aprovador, consultado, informado (RACI).

RAG

Veja geração aumentada via recuperação.

ransomware

Um software mal-intencionado desenvolvido para bloquear o acesso a um sistema ou dados de computador até que um pagamento seja feito.

Matriz RASCI

Veja responsável, aprovador, consultado, informado (RACI).

RCAC

Veja controle de acesso por linha e coluna.

réplica de leitura

Uma cópia de um banco de dados usada somente para leitura. É possível encaminhar consultas para a réplica de leitura e reduzir a carga no banco de dados principal.

Redefinir arquitetura

Veja 7 Rs.

objetivo de ponto de recuperação (RPO).

O máximo período de tempo aceitável desde o último ponto de recuperação de dados. Isso determina o que é considerado uma perda aceitável de dados entre o último ponto de recuperação e a interrupção do serviço.

objetivo de tempo de recuperação (RTO)

O máximo atraso aceitável entre a interrupção e a restauração do serviço.

refatorar

Veja 7 Rs.

Região

Uma coleção de AWS recursos em uma área geográfica. Cada um Região da AWS é isolado e independente dos outros para fornecer tolerância a falhas, estabilidade e resiliência. Para obter informações, consulte Specify which Regiões da AWS your account can use.

regressão

Uma técnica de ML que prevê um valor numérico. Por exemplo, para resolver o problema de “Por qual preço esta casa será vendida?” um modelo de ML pode usar um modelo de regressão linear para prever o preço de venda de uma casa com base em fatos conhecidos sobre a casa (por exemplo, a metragem quadrada).

redefinir a hospedagem

Veja 7 Rs.

versão

Em um processo de implantação, o ato de promover mudanças em um ambiente de produção.

realocar

Veja 7 Rs.

redefinir a plataforma

Veja 7 Rs.

recomprar

Veja 7 Rs.

resiliência

A capacidade de uma aplicação de resistir ou se recuperar de interrupções. Alta disponibilidade e recuperação de desastres são considerações comuns ao planejar a resiliência na Nuvem AWS. Para obter mais informações, consulte Nuvem AWS Resilience.

política baseada em recurso

Uma política associada a um recurso, como um bucket do Amazon S3, um endpoint ou uma chave de criptografia. Esse tipo de política especifica quais entidades principais têm acesso permitido, ações válidas e quaisquer outras condições que devem ser atendidas.

matriz responsável, accountable, consultada, informada (RACI)

Uma matriz que define as funções e responsabilidades de todas as partes envolvidas nas atividades de migração e nas operações de nuvem. O nome da matriz é derivado dos tipos de responsabilidade definidos na matriz: responsável (R), responsabilizável (A), consultado (C) e informado (I). O tipo de suporte (S) é opcional. Se você incluir suporte, a matriz será chamada de matriz RASCI e, se excluir, será chamada de matriz RACI.

controle responsivo

Um controle de segurança desenvolvido para conduzir a remediação de eventos adversos ou desvios em relação à linha de base de segurança. Para obter mais informações, consulte Controles responsivos em Como implementar controles de segurança na AWS.

reter

Veja 7 Rs.

Retirada

Veja 7 Rs.

Geração Aumentada de Recuperação (RAG)

Uma tecnologia de IA generativa em que um LLM faz referência a uma fonte de dados autorizada que está fora de suas fontes de dados de treinamento antes de gerar uma resposta. Por exemplo, um modelo RAG pode realizar uma pesquisa semântica na base de conhecimento ou nos dados personalizados de uma organização. Para obter mais informações, consulte O que é RAG (geração aumentada via recuperação)?.

alternância

O processo de atualizar periodicamente um segredo para dificultar o acesso de um invasor às credenciais.

controle de acesso por linha e coluna (RCAC)

O uso de expressões SQL básicas e flexíveis que tenham regras de acesso definidas. O RCAC consiste em permissões de linha e máscaras de coluna.

RPO

Veja objetivo de ponto de recuperação.

RTO

Veja objetivo de tempo de recuperação.

runbook

Um conjunto de procedimentos manuais ou automatizados necessários para realizar uma tarefa específica. Eles são normalmente criados para agilizar operações ou procedimentos repetitivos com altas taxas de erro.

S

SAML 2.0

Um padrão aberto que muitos provedores de identidade (IdPs) usam. Esse recurso permite o login único federado (SSO), para que os usuários possam fazer login Console de gerenciamento da AWS ou chamar as operações da AWS API sem que você precise criar um usuário no IAM

para todos em sua organização. Para obter mais informações sobre a federação baseada em SAML 2.0, consulte Sobre a federação baseada em SAML 2.0 na documentação do IAM.

SCADA

Veja controle de supervisão e aquisição de dados.

SCP

Veja política de controle de serviço.

secret

Em AWS Secrets Manager, informações confidenciais ou restritas, como uma senha ou credenciais de usuário, que você armazena de forma criptografada. Consiste no valor secreto e em seus metadados. O valor secreto pode ser binário, uma única string ou várias strings. Para obter mais informações, consulte What's in a Secrets Manager secret? na documentação do Secrets Manager.

segurança desde a concepção

Uma abordagem em engenharia de sistemas que leva em consideração a segurança em todo o processo de desenvolvimento.

controle de segurança

Uma barreira de proteção técnica ou administrativa que impede, detecta ou reduz a capacidade de uma ameaça explorar uma vulnerabilidade de segurança. Existem quatro tipos primários de controles de segurança: preventivos, detectivos, responsivos e proativos.

hardening da segurança

O processo de reduzir a superfície de ataque para torná-la mais resistente a ataques. Isso pode incluir ações como remover recursos que não são mais necessários, implementar a prática recomendada de segurança de conceder privilégios mínimos ou desativar recursos desnecessários em arquivos de configuração.

sistema de gerenciamento de eventos e informações de segurança (SIEM)

Ferramentas e serviços que combinam sistemas de gerenciamento de informações de segurança (SIM) e gerenciamento de eventos de segurança (SEM). Um sistema SIEM coleta, monitora e analisa dados de servidores, redes, dispositivos e outras fontes para detectar ameaças e violações de segurança e gerar alertas.

automação de resposta de segurança

Uma ação predefinida e programada projetada para responder ou remediar automaticamente um evento de segurança. Essas automações servem como controles de segurança responsivos ou detectivos que ajudam você a implementar as melhores práticas AWS de segurança. Exemplos de ações de resposta automatizada incluem a modificação de um grupo de segurança da VPC, a aplicação de patches em uma instância do Amazon EC2 ou a alternância de credenciais.

Criptografia do lado do servidor

Criptografia dos dados em seu destino, por AWS service (Serviço da AWS) quem os recebe.

política de controle de serviços (SCP)

Uma política que fornece controle centralizado sobre as permissões de todas as contas em uma organização em AWS Organizations. SCPs defina barreiras ou estabeleça limites nas ações que um administrador pode delegar a usuários ou funções. Você pode usar SCPs como listas de permissão ou listas de negação para especificar quais serviços ou ações são permitidos ou proibidos. Para obter mais informações, consulte Políticas de controle de serviço na AWS Organizations documentação.

service endpoint (endpoint de serviço)

O URL do ponto de entrada para um AWS service (Serviço da AWS). Você pode usar o endpoint para se conectar programaticamente ao serviço de destino. Para obter mais informações, consulte Endpoints do AWS service (Serviço da AWS) na Referência geral da AWS.

acordo de serviço (SLA)

Um acordo que esclarece o que uma equipe de TI promete fornecer aos clientes, como tempo de atividade e performance do serviço.

indicador de nível de serviço (SLI)

Uma avaliação de um aspecto de performance de um serviço, como taxa de erro, disponibilidade ou throughput.

objetivo de nível de serviço (SLO)

Uma métrica alvo que representa a integridade de um serviço, conforme avaliado por um indicador de nível de serviço.

modelo de responsabilidade compartilhada

Um modelo que descreve a responsabilidade com a qual você compartilha AWS pela segurança e conformidade na nuvem. AWS é responsável pela segurança da nuvem, enquanto você é responsável pela segurança na nuvem. Para obter mais informações, consulte o Modelo de responsabilidade compartilhada.

SIEM

Veja sistema de gerenciamento de eventos e informações de segurança.

ponto único de falha (SPOF)

Uma falha em um único componente crítico de uma aplicação que pode interromper o sistema.

SLA

Veja cacordo de serviço.

SLI

Veja indicador de nível de serviço.

SLO

Veja objetivo de nível de serviço.

split-and-seed modelo

Um padrão para escalar e acelerar projetos de modernização. À medida que novos recursos e lançamentos de produtos são definidos, a equipe principal se divide para criar novas equipes de produtos. Isso ajuda a escalar os recursos e os serviços da sua organização, melhora a produtividade do desenvolvedor e possibilita inovações rápidas. Para obter mais informações, consulte Phased approach to modernizing applications in the Nuvem AWS.

SPOF

Veja ponto único de falha.

esquema em estrela

Uma estrutura organizacional de banco de dados que usa uma grande tabela de fatos para armazenar dados transacionais ou medidos e usa uma ou mais tabelas dimensionais menores para armazenar atributos de dados. Essa estrutura foi projetada para ser usada em um data warehouse ou para fins de inteligência comercial.

padrão strangler fig

Uma abordagem à modernização de sistemas monolíticos que consiste em reescrever e substituir incrementalmente a funcionalidade do sistema até que o sistema herdado possa ser desativado. Esse padrão usa a analogia de uma videira que cresce e se torna uma árvore estabelecida e, eventualmente, supera e substitui sua hospedeira. O padrão foi apresentado por Martin Fowler como forma de gerenciar riscos ao reescrever sistemas monolíticos. Para ver um exemplo de como aplicar esse padrão, consulte Modernizar incrementalmente os serviços Web herdados do Microsoft ASP.NET (ASMX) usando contêineres e o Amazon API Gateway.

sub-rede

Um intervalo de endereços IP na VPC. Cada sub-rede fica alocada em uma única zona de disponibilidade.

controle supervisório e aquisição de dados (SCADA)

Na manufatura, um sistema que usa hardware e software para monitorar ativos físicos e operações de produção.

symmetric encryption (criptografia simétrica)

Um algoritmo de criptografia que usa a mesma chave para criptografar e descriptografar dados.

testes sintéticos

Testar um sistema de forma que simule as interações do usuário para detectar possíveis problemas ou monitorar a performance. Você pode usar o Amazon CloudWatch Synthetics para criar esses testes.

prompt do sistema

Uma técnica para fornecer contexto, instruções ou orientações a um LLM a fim de direcionar seu comportamento. Os prompts do sistema ajudam a definir o contexto e a estabelecer regras para interações com os usuários.

T

tags

Pares de valores-chave que atuam como metadados para organizar seus recursos. AWS As tags podem ajudar você a gerenciar, identificar, organizar, pesquisar e filtrar recursos da . Para obter mais informações, consulte Marcar seus recursos do AWS.

variável-alvo

O valor que você está tentando prever no ML supervisionado. Ela também é conhecida como variável de resultado. Por exemplo, em uma configuração de fabricação, a variável-alvo pode ser um defeito do produto.

lista de tarefas

Uma ferramenta usada para monitorar o progresso por meio de um runbook. Uma lista de tarefas contém uma visão geral do runbook e uma lista de tarefas gerais a serem concluídas. Para cada tarefa geral, ela inclui o tempo estimado necessário, o proprietário e o progresso.

ambiente de teste

Veja ambiente.

treinamento

O processo de fornecer dados para que seu modelo de ML aprenda. Os dados de treinamento devem conter a resposta correta. O algoritmo de aprendizado descobre padrões nos dados de treinamento que mapeiam os atributos dos dados de entrada no destino (a resposta que você deseja prever). Ele gera um modelo de ML que captura esses padrões. Você pode usar o modelo de ML para obter previsões de novos dados cujo destino você não conhece.

gateway de trânsito

Um hub de trânsito de rede que você pode usar para interconectar sua rede com VPCs a rede local. Para obter mais informações, consulte O que é um gateway de trânsito na AWS Transit Gateway documentação.

fluxo de trabalho baseado em troncos

Uma abordagem na qual os desenvolvedores criam e testam recursos localmente em uma ramificação de recursos e, em seguida, mesclam essas alterações na ramificação principal. A ramificação principal é então criada para os ambientes de desenvolvimento, pré-produção e produção, sequencialmente.

Acesso confiável

Conceder permissões a um serviço que você especifica para realizar tarefas em sua organização AWS Organizations e em suas contas em seu nome. O serviço confiável cria um perfil vinculado ao serviço em cada conta, quando esse perfil é necessário, para realizar tarefas de

gerenciamento para você. Para obter mais informações, consulte Usando AWS Organizations com outros AWS serviços na AWS Organizations documentação.

tuning (ajustar)

Alterar aspectos do processo de treinamento para melhorar a precisão do modelo de ML. Por exemplo, você pode treinar o modelo de ML gerando um conjunto de rótulos, adicionando rótulos e repetindo essas etapas várias vezes em configurações diferentes para otimizar o modelo.

equipe de duas pizzas

Uma pequena DevOps equipe que você pode alimentar com duas pizzas. Uma equipe de duas pizzas garante a melhor oportunidade possível de colaboração no desenvolvimento de software.

U

incerteza

Um conceito que se refere a informações imprecisas, incompletas ou desconhecidas que podem minar a confiabilidade dos modelos preditivos de ML. Há dois tipos de incertezas: a incerteza epistêmica é causada por dados limitados e incompletos, enquanto a incerteza aleatória é causada pelo ruído e pela aleatoriedade inerentes aos dados. Para obter mais informações, consulte o guia Como quantificar a incerteza em sistemas de aprendizado profundo.

tarefas indiferenciadas

Também conhecido como trabalho pesado, trabalho necessário para criar e operar um aplicativo, mas que não fornece valor direto ao usuário final nem oferece vantagem competitiva. Exemplos de tarefas indiferenciadas incluem aquisição, manutenção e planejamento de capacidade.

ambientes superiores

Veja ambiente.

V

aspiração

Uma operação de manutenção de banco de dados que envolve limpeza após atualizações incrementais para recuperar armazenamento e melhorar a performance.

controle de versões

Processos e ferramentas que rastreiam mudanças, como alterações no código-fonte em um repositório.

emparelhamento da VPC

Uma conexão entre duas VPCs que permite rotear o tráfego usando endereços IP privados. Para ter mais informações, consulte O que é emparelhamento de VPC? na documentação da Amazon VPC.

Vulnerabilidade

Uma falha de software ou hardware que compromete a segurança do sistema.

W

cache quente

Um cache de buffer que contém dados atuais e relevantes que são acessados com frequência. A instância do banco de dados pode ler do cache do buffer, o que é mais rápido do que ler da memória principal ou do disco.

dados mornos

Dados acessados raramente. Ao consultar esse tipo de dados, consultas moderadamente lentas geralmente são aceitáveis.

função de janela

Uma função SQL que executa um cálculo em um grupo de linhas que se relacionam de alguma forma com o registro atual. As funções de janela são úteis para processar tarefas, como calcular uma média móvel ou acessar o valor das linhas com base na posição relativa da linha atual.

workload

Uma coleção de códigos e recursos que geram valor empresarial, como uma aplicação voltada para o cliente ou um processo de backend.

workstreams

Grupos funcionais em um projeto de migração que são responsáveis por um conjunto específico de tarefas. Cada workstream é independente, mas oferece suporte aos outros workstreams do

projeto. Por exemplo, o workstream de portfólio é responsável por priorizar aplicações, planejar ondas e coletar metadados de migração. O workstream de portfólio entrega esses ativos ao workstream de migração, que então migra os servidores e as aplicações.

WORM

Veja gravação única e várias leituras.

WQF

Veja AWS Workload Qualification Framework.

gravação única e várias leituras (WORM)

Um modelo de armazenamento que grava dados uma única vez e evita que os dados sejam excluídos ou modificados. Os usuários autorizados podem ler os dados quantas vezes forem necessárias, mas não podem alterá-los. Essa infraestrutura de armazenamento de dados é considerada imutável.

Z

exploração de dia zero

Um ataque, normalmente malware, que tira proveito de uma vulnerabilidade zero-day.

vulnerabilidade de dia zero

Uma falha ou vulnerabilidade não mitigada em um sistema de produção. Os agentes de ameaças podem usar esse tipo de vulnerabilidade para atacar o sistema. Os desenvolvedores frequentemente ficam cientes da vulnerabilidade como resultado do ataque.

prompt zero shot

Fornecer a um LLM instruções para realizar uma tarefa, mas sem exemplos (shots) que possam ajudar a orientá-lo. O LLM deve usar seu conhecimento pré-treinado para lidar com a tarefa. A eficácia dos prompts zero-shot depende da complexidade da tarefa e da qualidade do prompt. Veja também prompts few-shot.

aplicação zumbi

Uma aplicação que tem um uso médio de CPU e memória inferior a 5%. Em um projeto de migração, é comum retirar essas aplicações.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.